Inside the Intel® Itanium® 2 Processor:

an Itanium Processor Family member for balanced performance over a wide range of applications

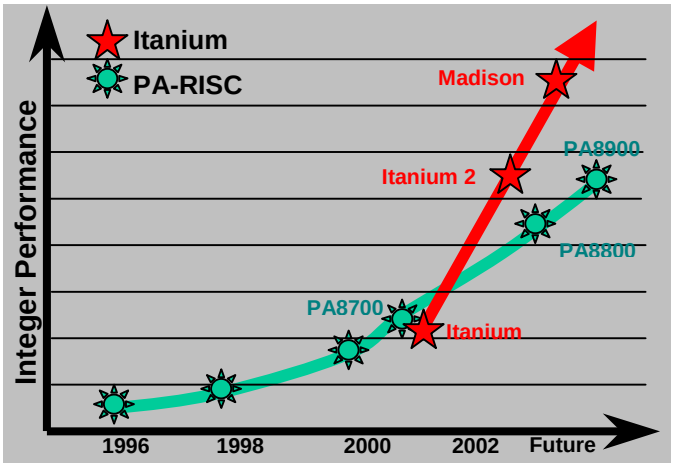
a Hewlett Packard
Technical White Paper
July 2002

invent

TABLE OF CONTENTS

The Itanium 2 processor, an inflection point for HP and the Industry

The Itanium 2 processor is the second product in the Itanium Processor Family (IPF). It is a joint effort between Hewlett-Packard and Intel. The Itanium 2 processor release is expected to create a significant inflection point in the computer industry moving from multiple unique proprietary RISC architectures to a new 21st century high performance architecture which, with the backing of many computer companies, will become an industry standard. HP and Intel entered into a partnership in 1994 to develop this new processor architecture. HP's involvement in the design of the Itanium 2 processor indicates the strong commitment HP has made to move to industry standard Itanium processor systems.



Customer Benefits from Industry Standard Processors

Customers can greatly benefit by the move to industry standard processors. As with standards for software interfaces (APIs, UNIX standards, etc.) and with hardware bus standards (PCI, AGP, SCSI, etc.), an industry adopted processor standard will bring more opportunities and choice to the customer. Already many system vendors have committed to deliver Itanium systems, including HP, IBM, SGI, NEC, Fujitsu-Siemens, Bull and Unisys.

1. More Operating System Options

The Itanium Processor Family has support for almost all major operating systems (OS). HP-UX is available from HP, a 64-bit version of Windows2000 is available from Microsoft and Linux is available from HP and other sources. HP has taken a "multi-OS" support model for the Itanium Processor Family, providing the customer what they need, whether it is HP-UX, Windows or Linux. Flexibility in the choice of OS provides fewer restrictions for selecting software applications—more applications are available to the customer since they are not restricted to one OS.

2. More Software Application Options

With the backing of HP and Intel, and the support of many other computer companies, the Itanium Processor Family will gain industry standard recognition. Itanium is Intel's strategic answer for migration to a 64-bit instruction set. The Aberdeen Group has predicted that the Itanium architecture "will become the volume leader for enterprise

applications…within five to seven years of its debut"[1]. Many major Independent Software Vendors (ISVs) have already made public commitments to support Itanium, such as Oracle, SAP, PeopleSoft and Microsoft SQL Server. Intel has even created an "IA64 Fund" to stimulate ISV porting to Itanium platforms.

Itanium platforms will become the preferred platform for software development when they are pervasive in the industry and when development tools are readily available. As a preferred development platform, new applications will be available *first* in the Itanium architecture. As a preferred development platform, new applications will be structured and performance tuned first for the Itanium architecture. The industry standard Itanium processors will provide the customer an abundant choice of software applications that are available early and are performance tuned.

3. More Platform Options

Currently multiple computer companies have indicated plans for providing Itanium platforms to their customers. Each computer company will seek to provide a unique value to the customer, allowing the customer to "shop around" for their optimal Itanium platform solution. Each platform is required to be compatible since they are powered by an Itanium processor from Intel, and must conform to a standard Itanium platform architecture. The competition between computer companies brings two major benefits to the customer: selection and competitive pricing.

HP Customer Software Support Throughout the Transition

Hewlett-Packard knows that the software transition from PA or IA-32 systems to Itanium processors will be an initial concern of some customers. HP (and others) will provide native and cross-platform compilers for a direct re-compilation into the native Itanium instruction set. These will provide the highest level of performance optimization.

Other codes may not be able to be recompiled (due to the source not being available) or there may be an immediate need to run existing PA or IA-32 binaries without a concern that it runs at optimal performance. For these cases, HP will provide two transition options:

1. PA-RISC Binary Translations

HP-UX systems sold for Itanium processors will have a special software package that will automatically detect PA-RISC application binaries and will automatically and transparently invoke a dynamic translator. The user is not required to initiate this process. The conversion from PA-RISC to Itanium instructions will be done using dynamic object code translation. The translator will also do some specific optimizations for the Itanium 2 microarchitecture.[1]

HP pioneered dynamic code translation technology, which was deployed in 1982 when PA-RISC was introduced, allowing HP customers to run their existing HP binaries on the new PA systems. This

[1] This translator will not work for OS or device driver codes.

same technology allows present-day customers to transparently run existing PA binaries unchanged on new Itanium Processor Family platforms.

2. IA-32 Binary Translations

Translation of IA-32 code is performed with special hardware built into the Itanium 2 processor. This hardware allows legacy I/O drivers and other lower performance codes to be translated and executed "on the fly" in the Itanium 2 processor.

HP-Intel Cooperation Creates a Computer Powerhouse!

The Itanium Processor Family is a joint effort of HP and Intel. Each company brings its strengths into the program.

- HP is known for its ability to design and deliver reliable high-performance processor systems.
- HP has years of experience in system design of large multiprocessor systems for commercial and technical applications.
- HP Labs first developed most of the fundamental concepts of the Itanium architecture.
- Intel has high-volume and world-class semiconductor manufacturing capabilities that provide highly reliable parts at a low cost.
- By being the only manufacturer of Itanium processors, Intel provides a level playing field for all computer system manufacturers to equally define and deliver Itanium platforms.

With the combined contributions of HP and Intel, the Itanium Processor Family will remain competitive well into the future.

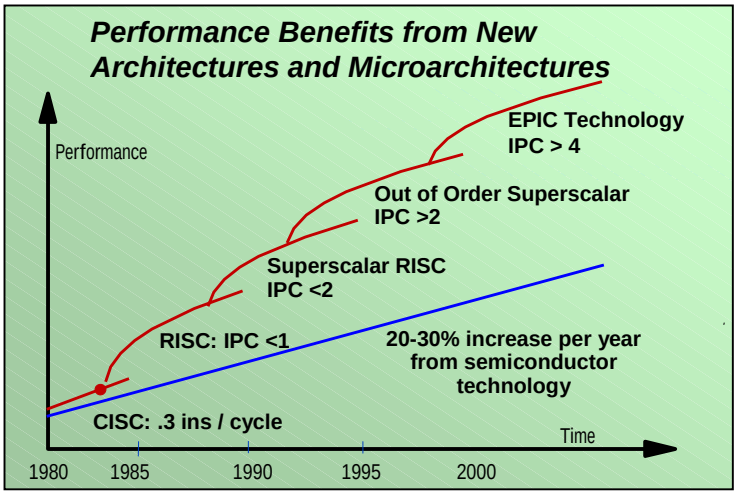
The first Itanium Processor Begins the Transition

Intel's original Itanium processor has provided the stimulus and needed Itanium system hardware to enable major efforts in software development. Work began to transition multiple operating systems and major applications to the Itanium platform. In parallel, firmware was written and validated on Itanium systems. System chipsets were developed and validated. The original Itanium processor demonstrated the feasibility of the new architecture, the Itanium instruction set, the system level architecture definitions, the firmware, the chipsets, and the OS and application software stacks.

The initial Itanium processor has enabled computer system suppliers to get ready for the Itanium architecture transition as well as to encourage many customers to begin to plan their transitions to Itanium systems. The Itanium 2 processor will benefit significantly from the momentum created by the initial Itanium release.

EPIC Technology: The Itanium Family Parallel Architecture

EPIC (Explicitly Parallel Instruction Computing) is the basis of the Itanium Processor Family architecture that is jointly owned by HP and Intel. EPIC was originally developed at HP Labs based on research dating back to 1989 [2]. The EPIC architecture is a new architecture advancement enabling processors to break through the limitations of RISC. This architecture gives compilers the ability to take advantage of parallel execution (Instruction Level Parallelism--ILP) beyond the limits of contemporary out-of-order (OOO) RISC designs.



EPIC designs move the complexity of Out-of-Order RISC from hardware to software.

Recent architectural inflection points have been brought about by the desire to reduce hardware complexity to help reduce the time it takes to design and verify new processors or to increase the processor frequency. RISC was the answer to reduce the complexity of CISC designs.

Microarchitectural techniques generally move architectures to a more complex implementation in order to improve performance without changing the instruction set architecture. Such is the case in the current out-of-order (OOO) RISC designs. The EPIC architecture is in part an answer to the increasing complexity of OOO RISC designs, and an attempt to reset the hardware to a less complex design point.

OOO RISC designs are very complex and require extensive time-consuming processor validation efforts. Current OOO RISC designs are hard to extend beyond 4 instructions wide and get justifiable performance improvements without adding multiple threads of execution in hardware. The EPIC designers recognized that if the compilers were given the right architectural support, then compilers could do much of what OOO hardware does today. These concepts will be discussed in detail later, but here are a few examples:

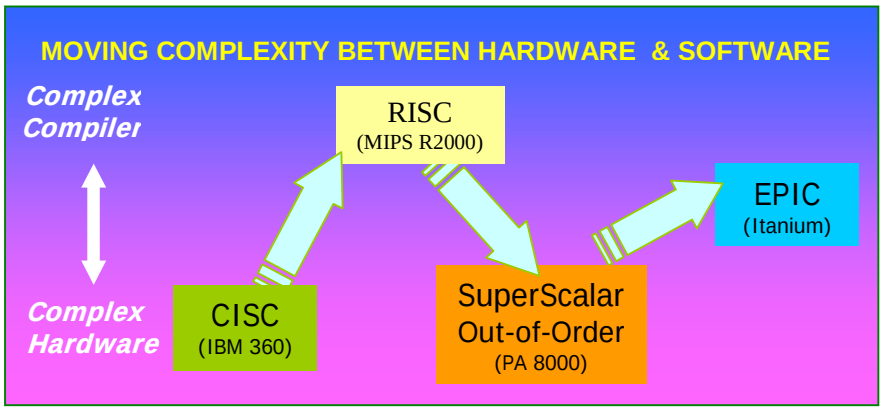
- EPIC allows compilers to define independent instruction sequences, which allows hardware to ignore dependency checks between these instructions. This same hardware functionality in OOO RISC designs is very costly and complex.

- Explicit larger integer and FP register files in the Itanium Processor Family replace the renaming registers and implicitly larger register files in OOO RISC designs. These explicit larger register files allow Itanium compilers to replace the functionality of the rename registers and to enable more operations to be executed in parallel because more architected registers are available.

- The predication function in EPIC allows explicit execution to proceed in parallel on both paths of a branch test, ignoring the results on the incorrect path when the branch test is resolved.

- In EPIC, loads and stores can be executed out-of-order due to control and data speculation features—in OOO RISC this must be done with special, complex (and hidden from the compiler) hardware. The Advanced Load Address Table (ALAT) in EPIC that is used for memory conflict resolution is smaller and simpler than the hardware required in typical OOO designs.

This does not mean to imply that EPIC hardware is simple. EPIC is not as simple as the original RISC designs. It is more correct to say that some of the complexity of OOO RISC designs that could be assigned to compiler-generated software has been transferred to the software. Other functions, which cannot be done in software, are required of the hardware. Some examples of the new hardware functions are the predicate bit controls, NaT bit controls, register file save and restore engine (RSE), and the hardware ALAT (Advanced Load Address Table). These will be discussed in the following sections. In summary, EPIC is a more "elegant" architecture because it does a better job of partitioning complexity and functionality between hardware and software.



The Itanium Processor Family requires advanced compilers for optimal performance. Fortunately this type of compiler technology has been in development at HP for many years and is now ready for commercial release.

Itanium Architecture Features

The Itanium architecture introduces many new features to enable high performance. Some of these are improvements in the integer register stack, in the branch architecture, in reducing memory latency, in multimedia processing, in software-pipelined loop execution and in technical floating-point processing.

1. Register File Improvements

With the popularity of structured and object oriented programming languages, procedure calls are becoming more frequent, and the required register context switches can amount to a significant portion of the processing time. The Itanium architecture provides two significant improvements to mitigate this problem.

Architected Integer Register File Sizes	
Itanium	128 registers
HP PA-RISC	32 registers
IBM POWER	32 registers
Sun UltraSparc	32 registers
Alpha	32 registers
IA-32	8 registers

- **Large physical integer register file stack.** Itanium architecture specifies 128 integer registers: 32 are "fixed" and 96 are "stacked". Each procedure call can allocate up to 96 of the "stacked" registers while having access to all 32 of the common "fixed" registers. Each procedure has its own register frame which is flexible in size (unlike the Sparc architecture). Most procedure calls only require a few new

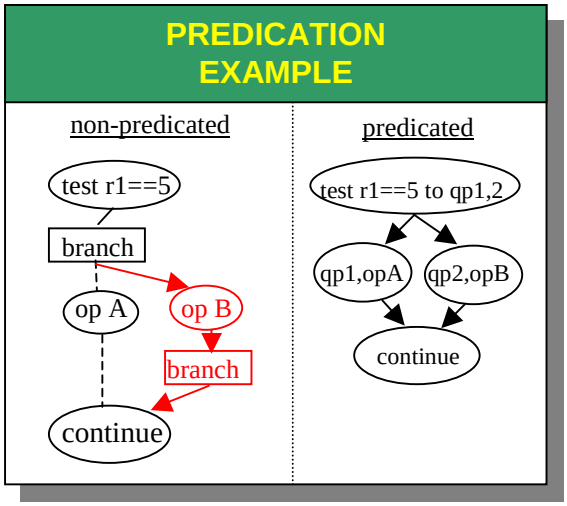
registers to be allocated, so many calls can be made before exceeding the physical limits of the register file. If the program returns without exceeding the physical register stack size, then no memory references are required to save and restore register contents. This is an effective method to reduce memory traffic and speed up procedure calls and returns.

- **Register Save Engine (RSE) in hardware.** The RSE hardware handles overflow and underflow conditions in the integer register file automatically. When a procedure call exceeds the number of available physical registers, this special hardware will free up register space by saving away older register frames into memory. When returning, the RSE hardware will automatically restore the proper register contents to the physical register file. These operations are transparent to the user and can be performed in a "lazy" (whenever required) or "eager" (opportunistic) way[2].

2. Branching Improvements

A major problem for all wide-issue (e.g. Alpha, PA-RISC) or hyper-pipelined (Pentium 4) microarchitectures is branching. Many branches can be reliably predicted and good branch prediction hardware can handle these branches, but some branches cannot be reliably predicted. Hard to predict branches are often mispredicted; if mispredicted, they are caught late in the execution pipeline and cause the processor to "re-steer" to a correct branch path. This re-steering causes invalidation of many instructions that are in-flight in the pipeline, and wastes execution resources for the time it takes to refill the pipeline. This can be a major loss of performance. The EPIC architecture provides several techniques to help reduce the impact of unpredictable branches.[3]

- **Predication.** Predication allows a compiler to eliminate an unpredictable branch. Both paths of a branch can be executed in parallel and the results from the correct path enabled with a single predicate bit. This is a compiler technique called *if-conversion*. The EPIC architecture provides for 63 addressable predicate bits, and these predicate bits control almost all EPIC instructions. In the illustration to the right, the normal coding involves a potentially mispredicted branch and a second branch to the continue block, but the predicated code removes both branches.



- **Early Branch Condition Testing.** The EPIC architecture separates out the test for a branch condition from the actual branch instruction. This feature allows the branch test to be done early to allow hardware to know the correct branch direction before the branch instruction actually generates a misprediction resteer.

- **Branch hinting.** The architecture provides a special branch hint instruction and allows branch hints on all branch instructions. These hints can indicate whether to use the branch taken (or untaken) information from software inputs (statically) or from prediction hardware (dynamically).

---

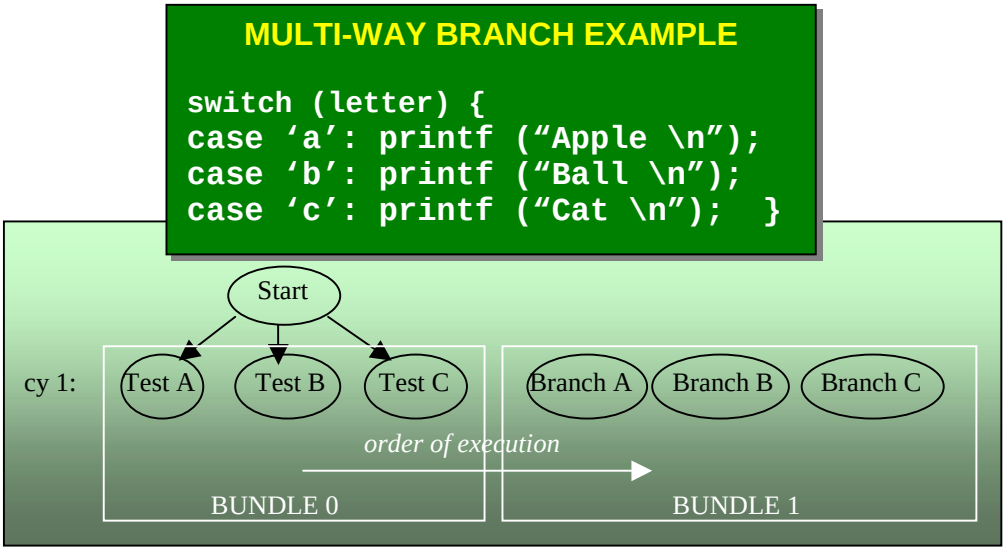
[2] Neither Itanium nor the Itanium 2 processors currently support eager RSE operations.

[3] This section describes branch prediction techniques that are optional for the hardware. The Itanium and Itanium 2 microarchitectures implement a subset of these techniques. *This section is not meant to imply that all techniques are implemented on either the Itanium or the Itanium 2 processors.*

If the branch is predicted taken, other hints can tell how far to prefetch along the predicted branch path, and how important it is to begin the prefetch (in case there are multiple prefetch requests). The branch hint instruction can provide the target address of an upcoming branch, allowing hardware to begin prefetching instructions from that target into the instruction caches. All of these hints, which can be provided by a good profiling compiler, allow the hardware to reduce branch mispredictions and effectively perform instruction prefetching.

- **Branch registers**. Indirect branching can be a performance problem because the target of the branch is usually unknown until the branch instruction is executed. The Itanium architecture defines eight 64-bit branch registers to allow the compiler to specify the branch targets before the branch instruction is reached. This provides a prefetch hint to the processor, allowing the processor to hide some or all of the latency of an indirect branch's potential instruction cache miss.

- **Multi-way branching**. A common branching dilemma is the execution of "switch" functions in C. A switch function tests one variable for several possible states and executes different code sequences for each. The Itanium architecture provides an effective way to implement multi-way branches. Using predication, multiple test conditions can be executed in parallel (up to six in one cycle for the Itanium 2 processor). The results of these tests can be used ***in the same issue group*** to control conditional branches based on those tests. Each EPIC bundle[4] of instructions can hold up to three predicated branches. Also because the execution order of the instructions in each bundle is explicitly architected, the predicate bits used to trigger the branch do not need to be fully qualified (some can be invalid, as long as there is a taken branch ahead of their use). Instead of serialized branch testing, the Itanium architecture allows the evaluation of multiple branches quickly by doing them in parallel. A three-condition switch can be executed in one processor cycle on the Itanium 2 microprocessor as illustrated below.



- **Full 64-bit IP relative branching via immediate field.** In EPIC full 64-bit branching is available without using the branch registers, by using the instruction's immediate field. This can be decoded by the instruction prefetching hardware early in the pipeline and used to trigger instruction prefetching.

[4] An EPIC instruction bundle holds three instructions with a special template tag defining the types of instructions and "execution stop" information. Stops will be discussed later.

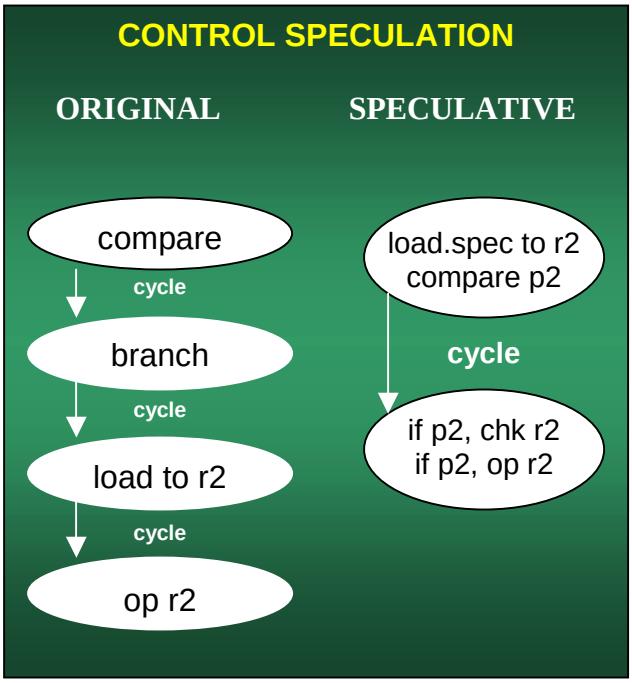
- **Loop branch control**. Software loops (“for”, “while”, “do”) can be determined by a fixed count known at the start of a loop or a condition tested every iteration of the loop. The Itanium architecture provides new branch instructions that allow the hardware to predict loop termination conditions early.

3. Memory Latency Reductions

Predication helps remove branches from the critical execution paths, but sooner or later the program will need to do loads from memory. Everyone would like all memory accesses to be one cycle, but due to real world constraints that’s just not possible. The first level cache hit on the Itanium 2 processor has a latency of only 1 cycle, but a third level cache hit takes 12 cycles, and system memory may take 100 or more cycles. A good architecture, microarchitecture and compiler can help lower the effective latency of a memory operation.

Scheduling code around memory load operations is difficult, because load latencies can be long and unpredictable. Moving the load ahead of branches and stores speculatively can effectively reduce memory latencies but is unsafe (due to possible page faults or load/store ordering violations) on many RISC architectures. The EPIC architecture allows these performance optimizations to be performed safely and efficiently and provides several other memory latency reduction techniques.

- **Control Speculation** allows loads to be executed early by moving them ahead of branches without the penalty of spurious faults. If a speculative access is made to a page of memory that is absent, invalid, or one the user is not permitted to access, a conventional architecture would require a costly fault to be taken. Using a *speculative load instruction*, Itanium processors defer a fault until the results of the load are actually used (determined to be non-speculative). A faulting speculative load result is given a NaT (not-a-thing) deferral token for integers, or a NaTVal is encoded in a floating-point operand. Most Itanium instructions will propagate an input NaT to their results, thus allowing a speculative code sequence to continue through multiple instructions correctly and efficiently. A simple example is illustrated to the right.

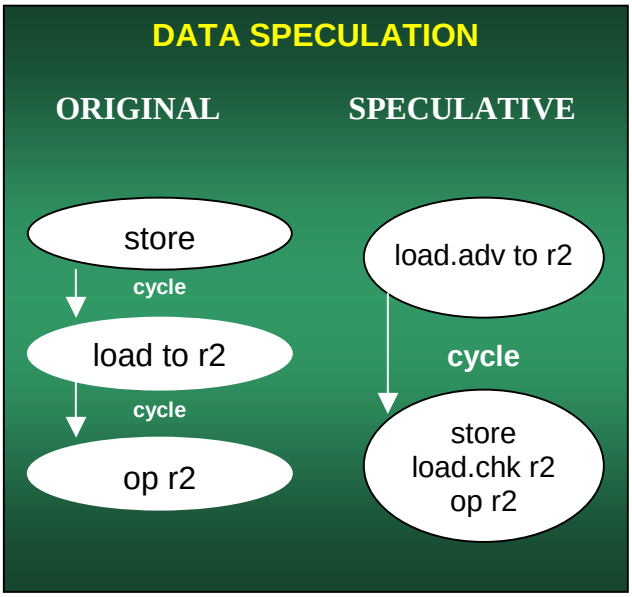


The result of a speculative operation must be checked by a special check instruction. This check instruction can be done in parallel with the use of the speculative result—it has no execution time penalty. When a speculative result is found to have a NaT or NaTVal token, a branch will be taken to recovery code provided by the compiler. The speculative operation will be re-executed and any faults that persist will be taken normally. Load speculation can be combined with predication, but stores are never performed speculatively.

An important use of control speculation is traversing a linked list while simultaneously checking for the NULL pointer. Accessing past the NULL pointer would normally cause a fault, but as a speculative load, the fault is deferred and is not taken if the data is never used.

- **Data Speculation** allows loads to be advanced safely before stores that may change the value of the load. In modern programming languages it is common to allow programs to use pointers to data types. Sometimes when using pointers it is hard for the compiler to determine if load and store operations are referencing the same memory location or not. To be safe, a compiler for a conventional architecture must not reorder a load above the store.

Using an advanced load (ld.a) instruction, Itanium processors can move loads ahead of stores, while tracking the integrity of the advanced load data until the load data is used. The advanced load works like a normal load except that it also maintains an entry in an on-chip Advanced Load Address Table (ALAT) with the load's register number, address and size. Any store to that address between the advanced load and a later check will invalidate the entry in the ALAT.



When the results of the advanced load are to be used in the code, a check instruction (ld.c or chk.a) is issued along with the use of the result of the advanced load. The check instruction does not have any execution latency. The check instruction queries the ALAT structure to make sure the advanced load's entry is still valid (i.e. a store to the same address hasn't been encountered between the advanced load and its use). If the ALAT indicates that the load data has been modified, then the load (and sometimes other code) is executed again, and the updated data value is loaded into the register file.

- **Explicit cache line prefetching** is another way to reduce effective memory latency. The Itanium instruction set defines a cache line fetch hint instruction which anticipates the use of cache data and brings in a cache line from memory with specific privileges (read only or write), and it can direct that cache line to a specific cache level. There are versions of this instruction that allow a fault (for example, a page fault) to be taken immediately or delayed until the cache line is actually used.

- **Cache hints** are provided in the EPIC architecture to optimize the use of multilevel cache structures. Hints are provided as to which cache level a cache line should be promoted, and how long is it expected to be used. If a cache line is classified as non-temporal (*.nta*), then the cache may choose to replace that cache line before others. These hints can help prevent polluting smaller first or second level caches with large data streams that are used briefly. In addition, they can be used to reduce the effective latency of a memory access by bringing data into the caches before the program needs them.

4. Multimedia Functionality

The Itanium architecture includes modern multimedia processing and floating-point streaming instructions needed by many types of multimedia or scientific processing. Multimedia applications are growing, and include streaming video, animation, 2D and 3D image generation and rendering, image processing, audio and video compression/decompression, music and voice synthesis, voice recognition, games and video-conferencing. Multimedia processing involves decoding, encoding, interpreting, enhancing and rendering of digital data. The instructions provided in the EPIC architecture are similar to HP PA-RISC's MAX-2 and Intel's IA-32 MMX and SSE instructions.

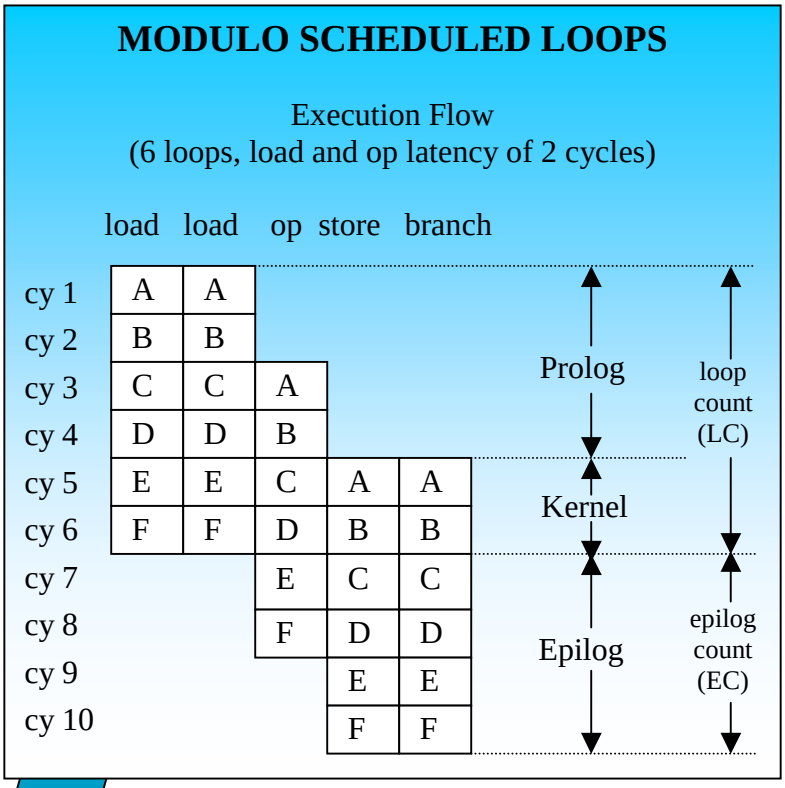
- **Parallel 32-bit integer multimedia instruction support**. Parallel arithmetic instructions are provided for 32, 16 and 8 bit data. A performance improvement of 2x or more can be obtained by using these instructions when data sets are less than 64 bits. Operations may optionally be performed with saturating arithmetic.

- **Streaming SIMD floating-point extensions (SSE)**. The Itanium architecture defines parallel single-precision floating-point instructions that can effectively double the performance for this type of data operation.

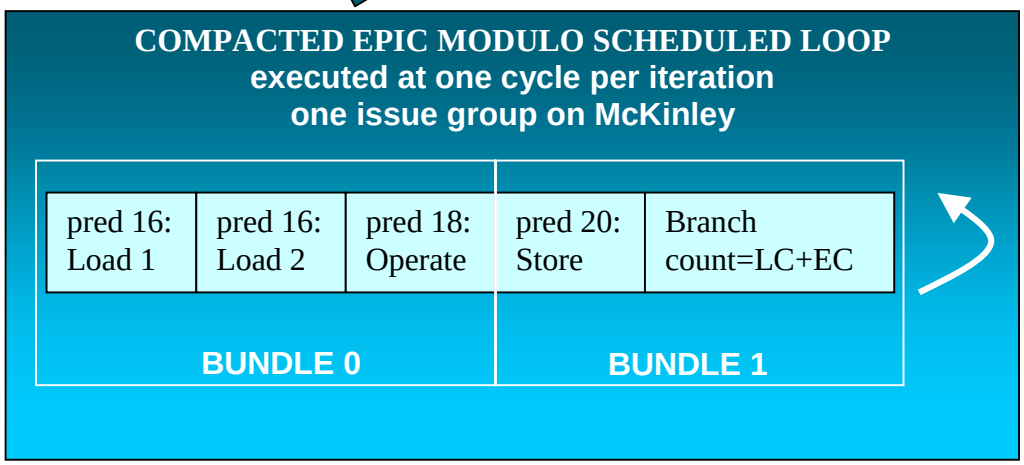
- **I/O Line Coalescing.** I/O operations for multimedia applications are normally performed in 32-bit or 64-bit chunks and are generally not written to cacheable memory space. If each operation is sent directly to the system bus for transmission to the I/O device, and due to the slower data rates on a system bus, performance would suffer greatly. The Itanium architecture provides for special coalescing memory pages that hint to the processor hardware to coalesce these store operations into cache line sized chunks before writing the data to the I/O device. Other instructions are provided to force partial lines to be written out of the processor. Load requests may also be combined into a single larger bus transaction and loaded into the coalescing buffer.

5. Modulo Scheduled Software-pipelined Loops

The EPIC architecture has provided an efficient method for modulo-scheduling loops with very little overhead. Because there is little overhead, this technique can be effectively applied to large or small loops, providing more opportunities for compiler optimizations. A basic load > load > operate > store > branch loop is illustrated. (This is typical of many floating-point loops). The execution is broken into three phases: prolog, kernel, and epilog. The EPIC architecture allows all three phases to be coded into a single compact loop controlled by predicate bits.



The EPIC implementation of modulo scheduled loops pulls together three main architectural techniques: rotating registers, rotating predicates, and loop branch control. The rotating registers automatically increment the register number of the physical register stack by a specified increment each iteration of the loop, assuring that the next loop will not overwrite previous data. The rotating predicates enable operations to phase-in during the prolog and phase-out during the epilog. The loop branch support repeats the execution until all iterations are completed and can perfectly predict the exit out of the loop. The user or compiler provides two values for the branch control of loops. The loop count (LC) is the number of iterations for the prolog and kernel phases of execution. The epilog count (EC) indicates the iterations needed to finish up the remaining in-progress operations.



6. Floating-Point Intensive Technical Applications

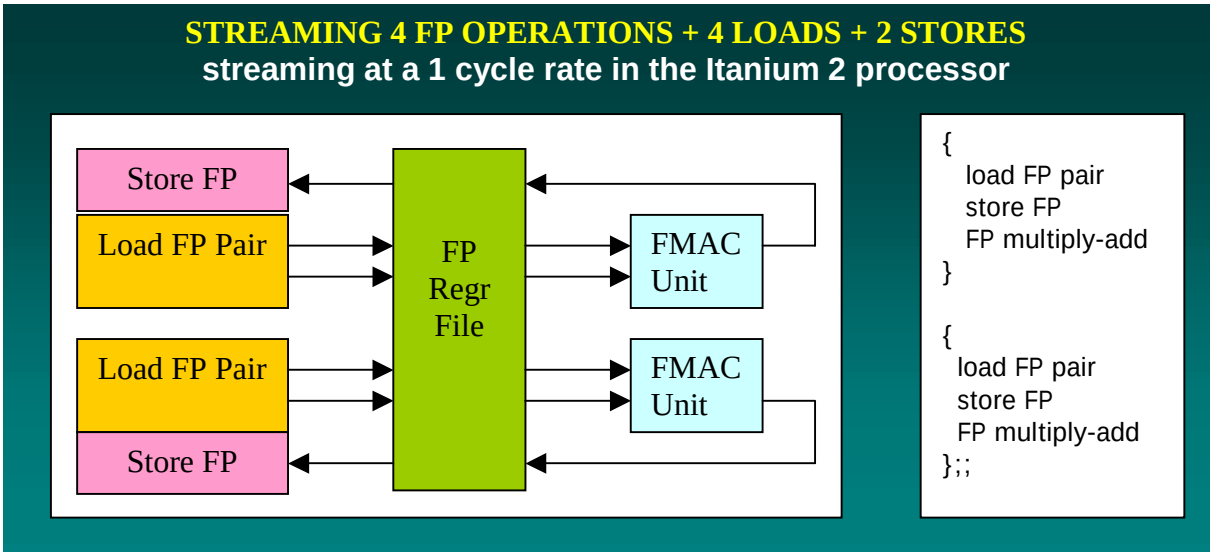
Technical applications can benefit from many of the features already discussed—software pipelining of loops, predication, speculation, SIMD streaming instructions, and the large integer and floating-point register files. Some additional architectural features are worth mentioning.

- **Large FP Register File.** The architecture defines 32 "fixed" and 96 "rotating" FP registers. These rotating registers are used by the software pipelining hardware to provide automatic loop pipelining. (Rotating registers are provided in the integer register file to support software pipelining too.)

- **Load FP Pair Instruction.** The floating-point load pair instruction is a very efficient way to enable streaming FP data to the FP register file and FP execution units. With two FP load pair instructions, the input operands for two FP multiply-add instructions can be supplied using only two instructions of the 6 issued each cycle on the Itanium 2 processor.

- **Enhanced FP Accuracy.** The floating-point units are 82 bits wide. With 82 bits of precision the FP hardware can support single, double and 80-bit double-extended IEEE standard values.

- **MMF bundle.** The MMF (**M**emory op, **M**emory op, **F**loating-point op) instruction bundle is of special value to technical FP inner loops of the DAXPY[5] variety. This template, in combination with the load FP pair instruction, allows the construction of tight and efficient floating-point inner loops. For example, by executing two MMF bundles in a single cycle, the Itanium 2 processor can supply the load and store bandwidth to execute 4 FLOPs per cycle (illustrated below)[6].



[5] DAXPY is a double-precision floating-point operation of the form A * X + Y, which is commonly found in the inner codes of many technical applications.

[6] Streaming at a 1 cycle rate can be supported when accessing data from the L2 cache.

Flexible Parallelism Without Recompiling

The Itanium and the Itanium 2 microprocessors are designed to support a sustained issue rate of 6 instructions (or 2 bundles) per cycle. If someone wanted to build a low cost, low power Itanium processor that issues only one instruction per cycle, or a high-end Itanium processor that issues 12 instructions per cycle—would that require re-compilation? Re-compilation was required in early VLIW architectures, but *not* with the EPIC architecture.

Variable Instruction Group Examples

```
M;;
MI;;
MMI MII MFI MII M;;
MFI MFI MFI MFI;;
```

where *M= memory operation*
I = integer operation
F = floating-point op

The EPIC architecture allows the compiler to define the maximum amount of instruction-level parallelism (the maximum number of instructions that can be issued together) independent of the number of instruction groups (bundles) a particular processor is capable of issuing. For example the EPIC compiler could emit 128 (or more) instructions that could be executed in parallel, allowing the hardware to be effectively restricted only by its own resources. The Itanium 2 processor would execute these 128 instructions in groups of 6 instructions, but another hardware implementation may issue them in groups of, for example, 1, 3 or more.

Central to the ability of the EPIC architecture to express instruction-level parallelism is the "stop". (A stop is indicated by ";;" in the assembly code.) The instructions contained between stops are called an instruction group, because these instructions can all be issued in parallel and be guaranteed to be free of interdependencies. The architecture defines some templates (templates are patterns of 3 instructions within a bundle) with intra-bundle stops to enable efficient instruction packaging when instruction parallelism is limited. In addition to intra-bundle stops, the EPIC architecture templates provide each bundle with the option of a stop following the last instruction, allowing the compiler to string multiple bundles together into one large instruction group if no stops are present. This flexibility allows an instruction group to range in size from less than a bundle up to many bundles.

Types of Stops

Intra-Bundle Stops

```
MI;;I
M;;MI
```

End-of-bundle Stop

Template Definition

Instruction 2	Instruction 1	Instruction 0	Template	**Stop**

Addressing Concerns about the EPIC Architecture

Since the announcement of the EPIC architecture, some in the computer industry (mainly competitors!) have voiced objections to this new architecture. The most frequently voiced concerns about EPIC are addressed below.

- ✓ **"… but what about code bloat?"** The Itanium instruction set will have a larger instruction "footprint" than many of its competitors for several reasons:

  - an Itanium instruction is 42.6 bits in size (128 bits per bundle/3 instructions) versus other RISCs at 32 bits
  - bundling of instructions into specific template types can lead to some wasted instructions (bundle padding with nops)
  - techniques of speculation and prefetching can require more instructions to be generated (e.g. a speculative request followed by a check instruction)
  - base+displacement addressing is not provided, sometimes generating an addition instruction

  For these reasons the Itanium code size is larger than many competitors, but there are several reasons why this is not a large concern for Itanium processors.

  + On-chip memory is getting cheaper, and on the Itanium 2 processor there is up to 3 ¼ megabytes of cache on chip. Future processors are expected to grow their on-chip caches. So an increase in instruction footprint is offset by increased cache sizes on the processor.
  + The Itanium and Itanium 2 processors both have three level cache structures. An L1i (instruction cache) miss is backed up by a fast access to the next level cache. In the case of the Itanium 2 processor, the next level cache is a 256-kilobyte cache with a 7-cycle latency for instructions. This L2 cache is much larger than most instruction caches which can compensate for the larger instruction cache demands of EPIC.
  + Common arithmetic operations, called integer ALU instructions[7], can be issued in either M or I instruction slots[8]. This makes the EPIC template options more flexible for common functions.
  + Special templates have been architected which provide a stop in the middle of a bundle, for use when instruction parallelism does not extend throughout the bundle. These special templates help reduce "code bloat" by more effectively using each instruction slot.
  + The register save engine (RSE) defined in EPIC actually reduces the number of instruction that are explicitly specified by making the register save and restore function an implicit hardware function.
  + The modulo loop scheduling features of EPIC can reduce the instruction footprint of highly tuned kernels by reducing the amount of loop unrolling that is needed to attain high performance and by eliminating the need for separate sections of code for the three phases of execution (prolog, kernel, epilog).
  + Some instructions, such as the load floating point pair and multimedia instructions actually cut in half the number of instructions needed for some operations.

[7] EPIC groups all instructions into six instruction types: A (integer ALU), I (non-ALU integers), M (memory), F (floating-point), B (branch) and L+X (extended instructions).
[8] Each instruction bundle holds two or three instructions as defined by the bundle template. The template can identify each instruction slot as M (memory), I (integer), B (branch), F (floating-point) or L+X (extended instruction).

- + Most memory instructions provide a post-increment capability to eliminate the need for some additional address manipulation instructions.
- + Branch prediction and prefetching instructions are provided to efficiently stream code in from higher-level caches. Combined with predication, these code streams can be highly predictable. Prefetching these highly predictable instruction streams can minimize any ill effects of code size.

✓ **"… but there isn't enough instruction level parallelism for a parallel architecture"**. Some argue that there isn't enough instruction parallelism in typical applications to justify an architecture that is easily expandable to larger issue widths.

- + Studies on RISC processors indicate a reduced performance gain for increasing the issue rate beyond 4 instructions. This is true as far as it goes, but with the features provided in EPIC, parallelism can more easily be exposed and realized. As compilers expose more parallelism, more speculative operations are performed which require more processor resources. The increased performance can come at the cost of needing a wider issue width to support the additional processing required. This flexibility in going wider to gain additional performance is a feature of the EPIC architecture.
- + Not all applications will have a high level of instruction level parallelism (ILP), but some will. EPIC can enable hardware that will effectively execute applications with high ILP, but on applications with low ILP, other techniques such as simultaneous multithreading or software threads used to monitor and "tune-up" hardware performance can be used to utilize unneeded processor resources.

✓ **"… but memory latency is the problem in processor performance, not ILP".** Some have argued that the issue in today's high-end server achieving better performance requires improving memory latency more than ILP. Memory latency *is* a major issue for some applications like transaction processing. Some improvements can be made apart from architectural changes, such as providing larger caches and faster memory systems. EPIC provides architectural enhancements that can further decrease the effective memory latency for important applications such as transaction processing.

Some EPIC architectural features to help reduce memory latency are:

- + **Control Speculation—**starting memory requests early without penalty for faults or traps if the results are never used.
- + **Data Speculation—**advancing loads before stores to reduce their effective latency.
- + **Explicit Data Cache Prefetch instruction—**When data is suspected to reside outside of the cache closest to the processor, these instructions can bring the data into the cache before the program needs it.
- + **Cache hints—**These hint fields in memory instructions can manage placement and allocation of cache lines in a multi-level cache hierarchy.
- + **Instruction Prefetching—**several hints for instruction prefetching are provided in the IPF architecture, and the hardware can take advantage of them to reduce instruction fetch latency.
- + **Large Architected Register File—**the large register files architected on the Itanium Processor Family eliminates many memory operations because temporary values can be retained in the register file, not in memory. This also reduces thrashing of data in the first level data cache that can cause data to be removed prematurely.

These features, combined with software performing static and dynamic code optimization, can be used and fine-tuned to achieve an effective lower latency for memory requests.

- ✓ **"… but large register files increase context switch overhead"**. The Itanium instruction set specifies 128 integer and floating-point register files. Of these 128 registers, 32 are fixed and the rest are register stack or rotating. While a full context switch overhead can be higher in some cases, the following features mitigate this penalty.

  - **+** Users specify the size of each procedure's register frame, so not all of the 128 registers must be saved away on each context switch.
  - **+** The Register Save Engine (RSE) feature of EPIC provides for hardware to automatically do saves and restores to reduce context switch overhead. The "eager" RSE mode can automatically do "early" saves and restores as a background processing function.
  - **+** For the 128 floating-point registers, special user mask bits indicate if the upper or lower portions of the FP register stack have been used. This prevents saves and restores when the contents of the upper or lower FP registers have not been modified.

- ✓ **"… but EPIC is too complex; we need smaller, simpler processors".** EPIC is a very advanced architecture, the result of many decades of computer architecture research and development. Many applications need all of the features of EPIC to obtain new levels of performance. Applications needing the fastest single processor performance benefit from the instruction parallelism techniques, speculation and prediction that EPIC provides. Transaction processing can benefit from the many cache management techniques provided in EPIC as well as the large register file, speculation and prediction.

  EPIC can be used on small and simple processor implementations, if that is what an application needs. Simple Itanium processors can be built by reducing the issue width down to a single instruction. Some features of the architecture are optional for a given hardware implementation, since software can provide the missing functionality for low occurrence operations. The hardware Advanced Load Address Table (ALAT) for data speculation is optional for the hardware to implement. Even a simpler hardware implementation can benefit from the EPIC architecture because it puts more out-of-order complexity found in many RISC designs on the shoulders of the compiler, not on the hardware.

Future Expandability with New Hardware and Software Techniques

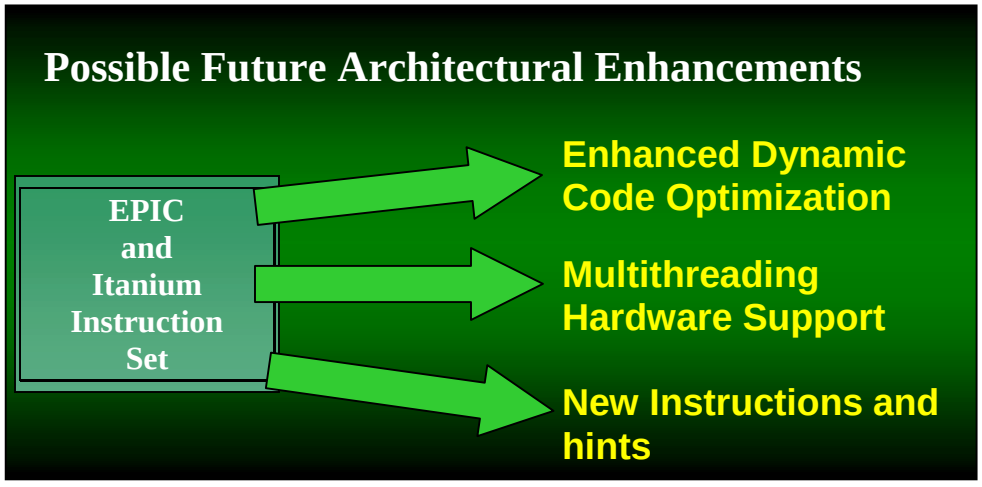
The EPIC architecture as implemented in the Itanium Processor Family Instruction Set is robust with performance features that can be exploited by future software and hardware developments as well as room to further enhance the ISA (Instruction Set Architecture). Some examples of possible future developments are:

- **Large instructions allow for easy enhancements.** The Itanium instruction set has many unused encodings that allow for the addition of new instructions and hints. These new instructions and hints can provide opportunities for increasing processor performance. Many RISC processors use only 32 bits for each instruction, which makes adding new instructions difficult, if not impossible.

- **Dynamic (run-time) code optimization**. Many performance hints are already provided in the architecture for improving branch prediction, cache prediction, instruction prefetching, data prefetching and data cache allocation of information. While an application is running, performance monitoring hardware can track performance issues such as cache misses or mispredicted branches. This information can be used today by optimizing software to decrease or remove these potential performance inhibitors.

  While the tools needed for dynamic optimization are currently available, new performance optimizations (which are currently not in the architecture) can be added later as additional hints to provide further performance tuning. New hints in the architecture are backward compatible because hints do not change the functional results, only performance.

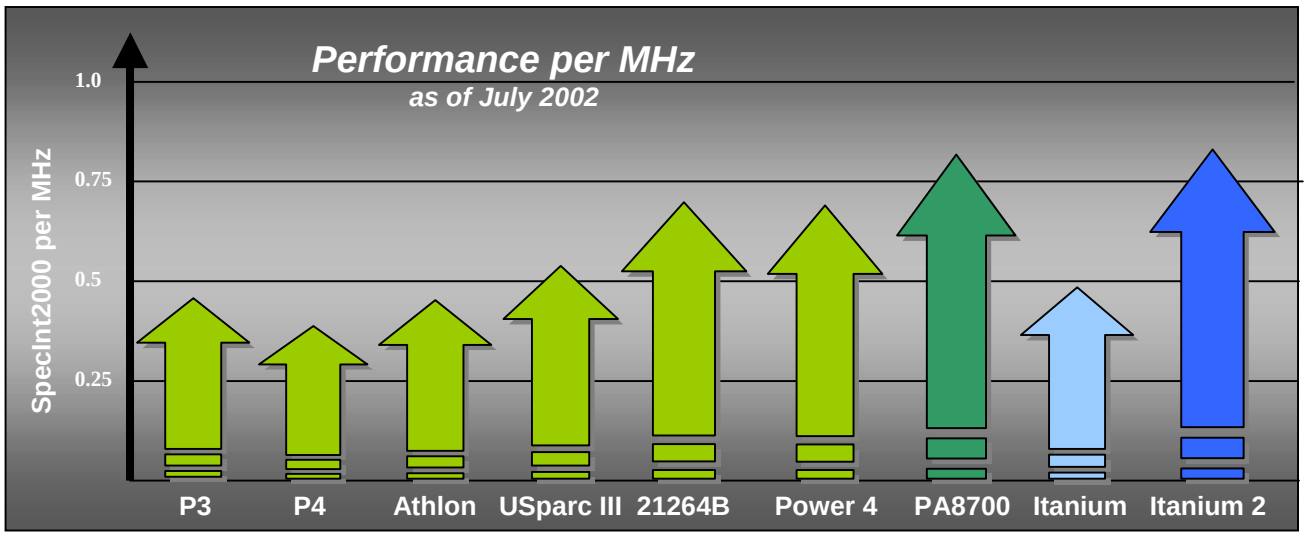
- **Multithreading support.** There are many types of hardware multithreading approaches that are potential enhancements to the current Itanium processor family. Simultaneous multithreading (two or more threads executing in parallel), alternating multithreading (threads alternating time-slices in the processor) or temporal or switch-on-event multithreading (switching on a specific performance condition) are all possibilities for future Itanium processors. The current EPIC architecture and Itanium Instruction Set, as is, or with some minor enhancements, can meet the needs of all of these multithreading microarchitectures.



The Itanium 2 Microarchitecture Distinctives

Computer performance is a result of many contributions--compiler, application and OS optimizations on the software side, and architecture, microarchitecture, processor frequency and system design on the hardware side. The Itanium 2 processor and the first Itanium processor were designed in parallel—not leveraged from the first Itanium to the second generation. Since the Itanium and Itanium 2 processors are the only Itanium architecture processor implementations at this time, the differences between them in microarchitecture will be highlighted below. As will be shown, these differences provide the Itanium 2 processor with more performance per MHz of processor frequency. The reasons for the increased performance on the Itanium 2 processor are detailed in this section.

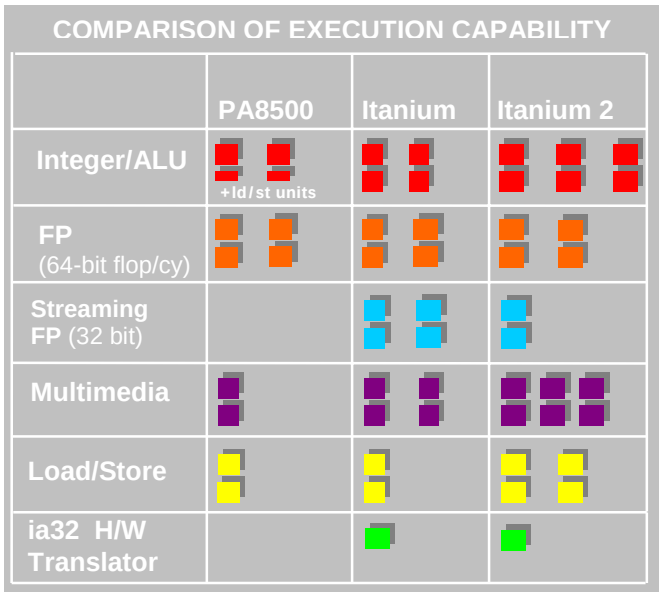
Some believe that processor frequency is the major contribution hardware can provide to improve performance. As a result, some processors implement deep pipelines which do smaller amounts of work each stage in order to increase the processor frequency. This technique can actually lower the effective performance per MHz of a processor because of the increased branch penalties for mispredicted branches and the overhead for additional pipeline registers and control logic. Another effect of deep pipelining is that it drives the processor frequency unnaturally higher causing increased processor power and heat problems. The Itanium 2 processor design is an attempt to balance microarchitecture and processor frequency to provide high performance at a lower processor frequency.



More Execution Resources

The Itanium 2 microarchitecture is designed to sustain the execution of 6 instructions per cycle (which equals 2 bundles per cycle in the Itanium architecture). Instruction fetching and prefetching bandwidth from the L2 cache provides 6 instructions per cycle to the L1 instruction cache. The L1 instruction cache can supply 6 instructions per cycle to the execution pipeline. The instruction decode, register file accesses and retirement logic all support 6 instructions per cycle execution. Note that the execution phase of the Itanium processor was usually not able to effectively execute 6 instructions per cycle because of limited execution resources, whereas the Itanium 2 microarchitecture provides several more execution resources to improve execution efficiency.

The Itanium architecture defines what type of instructions can be executed in two bundles (6 instructions). Two bundles can include up to 4 memory operations, up to 6 ALU operations, up to 4 integer operations, up to 6 branch operations, and up to 2 FP operations. The Itanium 2 processor can issue nearly all combinations of 6 instructions each cycle. Compared to the original Itanium processor, the Itanium 2 processor *has 2 additional ALU units, 2 additional multimedia units, and 2 additional load/store memory ports.*



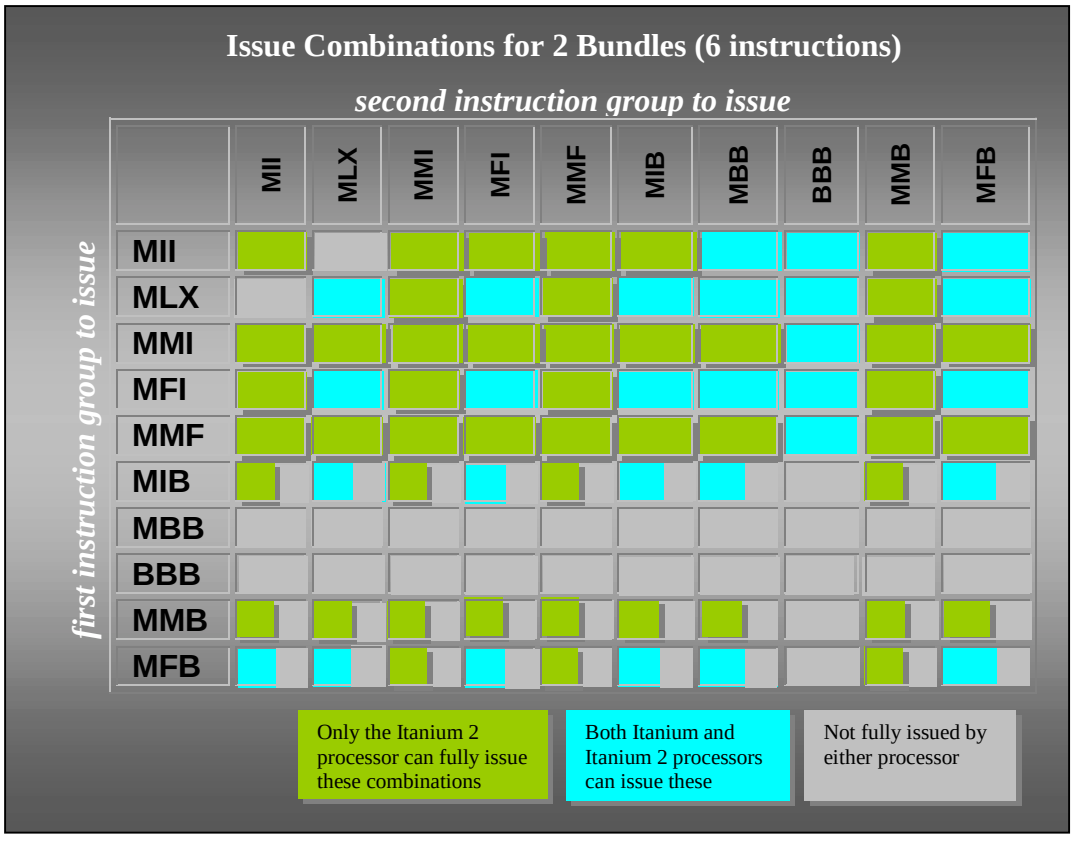
Itanium 2 Proc. Execution Units	# Units	Latency
Memory Load Pipes	2	1 cycle (L1)
Memory Store Pipes	2	NA
ALUs (integer)	6	1 cycle
Integer Units	2	1 cycle
Integer Shift	1	1 cycle
Multimedia ALUs	6	2 cycles
Parallel Multiply Units	1	2 cycles
Parallel Shift-Mask Units	2	2 cycles
Pop count	1	2 cycles
FP FMAC (multiply-accumulate)	2	4 cycles
FP FMISC (compares, merge, etc)	2	4 cycles
Branch Unit	3	0-2 cycles
IA-32 Engine	1	variable

The Itanium 2 processor has enough issue control and data paths to issue 11 instructions per cycle (Itanium can issue 9). While it would seem that no more than 6 are needed, having 11 issue ports allows ports to be dedicated to specific functions and to increase instruction dispersal efficiency. The Itanium 2 processor's issue ports are 4 memory/ALU ports, 2 integer/ALU ports, 2 floating-point ports, and 3 branch ports.

Better Issue Efficiency

Another way to visualize the increased resources and capability of the Itanium 2 processor is to understand the combinations of instruction bundles that **cannot** be fully issued in one cycle on the original Itanium processor and how they are now supported on the Itanium 2 processor. The chart below illustrates the types of instruction bundles that could be issued together (the rows indicate the first bundle of the pair, and the columns indicate the second bundle of the pair to be executed together). The squares colored gray cannot be issued together in one cycle on either the Itanium or Itanium 2 processors. The squares colored blue show combinations that can be issued in one cycle on both the Itanium and Itanium 2 processors. The squares colored green illustrates the additional execution capability of the Itanium 2 processor. Partially colored boxes indicate that if the first bundle group has only branch hint "B" instructions, then the second instruction bundle may be executed in parallel. If the first bundle contains true branch instructions, the second bundle cannot be executed in parallel.

Note that using this static analysis, the Itanium 2 processor has a 75% efficiency for 2-bundle (6 instruction) issues, compared to 28% for the original Itanium processor.



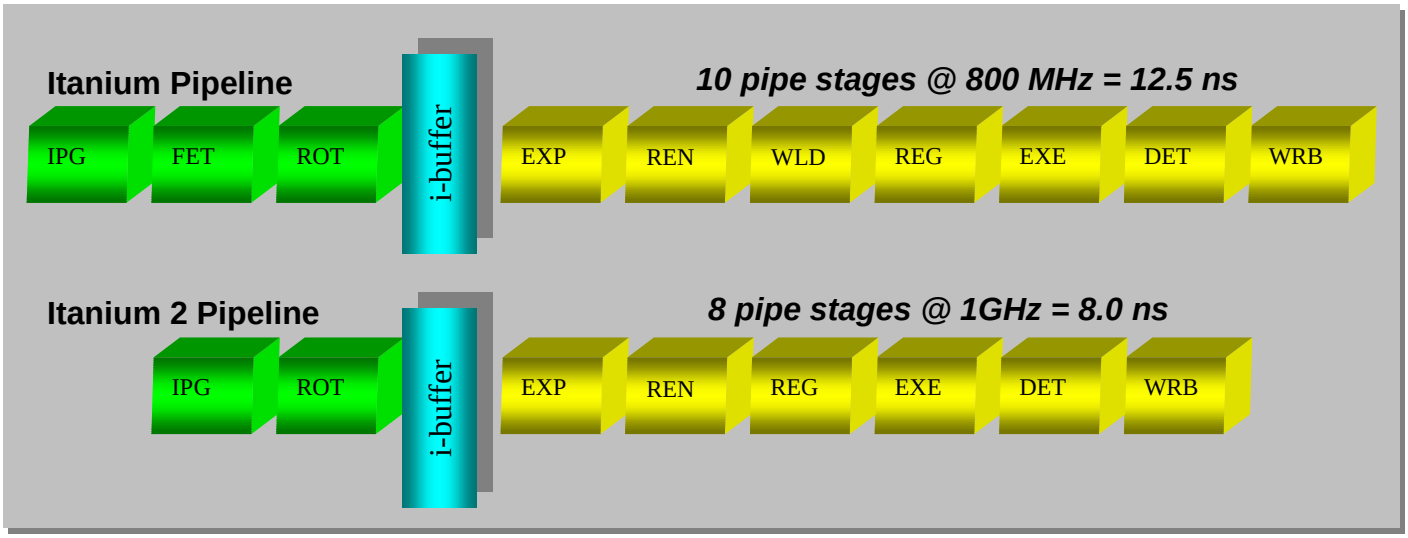
where M = memory op, I = integer op, F = floating-point op, B = branch op, and LX = long extension (for integer or branching)

Pipeline Enhancements

The Itanium 2 processor pipeline is 8 stages long, with an instruction buffer that decouples the "front-end" instruction fetching from the "back-end" execution stages. The Itanium 2 pipeline is two stages shorter than the original Itanium processor pipeline (see diagram below). It is estimated that the reduction of two pipeline stages increases performance by 4 to 6%. The primary benefit comes from shorter branch misprediction penalties—it takes less time to refill a shorter pipeline after a mispredicted branch is encountered.

The front-end pipe stage reduction is the result of a one-cycle instruction cache which allows the instructions and control information used to generate the next instruction address to be read out in one cycle. In a single pipeline stage, the Itanium 2 processor performs a read from the L1i and generates the IP (instruction pointer) needed for the next cycle's instruction fetch.

The WLD stage in Itanium was not needed in the Itanium 2 processor because of more aggressive circuit design techniques used in the register file design.



The Itanium 2 processor pipeline is an in-order issue pipeline with pipe stalls that immediately stop the stalled pipe stage and all preceding pipe stages. All instructions are completed in-order with respect to exceptions, providing precise exception handling. A scoreboard for integer, floating-point, and predicate registers is provided to allow out-of-order completion of results.

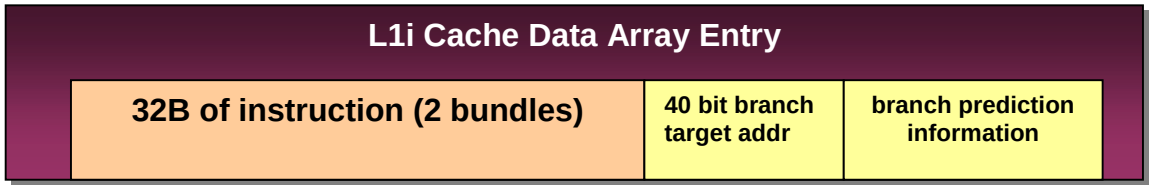
High Bandwidth Register Files

To support the demands of the 2-bundle execution pipeline, the Itanium 2 processor integer register file provides 12 read ports and 8 write ports. There are 128 physical registers (plus 16 banked registers) in the integer register file. Each entry is 65 bits, 64 bits of data and a single NaT bit. Each register access is completed in a single cycle.

The floating-point register file has 128, 82 bit entries. The floating-point register file has 8 read ports (6 source operands and 2 store data reads) and 6 write ports (2 FP results and 4 FP load returns).

Improved Branching

The shorter Itanium 2 processor pipeline improves branch performance (vs. the original Itanium processor). In addition, the Itanium 2 processor designers have done even more to improve branching. The L1 instruction cache uses a unique approach by combining the next address generation and first level branch prediction with the instruction data (see diagram below). This allows fast next address generation as well as early branch prediction. This "0 cycle branch re-steer" is a critical performance advantage for the Itanium 2 processor.



The Itanium 2 processor uses several branch prediction structures to improve branch prediction:

- **Branch History Table** – consists of a 2 level, 4 bit history algorithm, with the first level contained in the L1i cache, and the second level contained in a separate structure with 12K 4-bit histories.
- **Pattern History Table** – consists of 16K 2-bit counters
- **Return Stack Buffer** – 8 entries
- **Indirect Target Prediction** – uses 8 branch registers
- **Perfect Loop Prediction logic** – to calculate exits for looping codes.

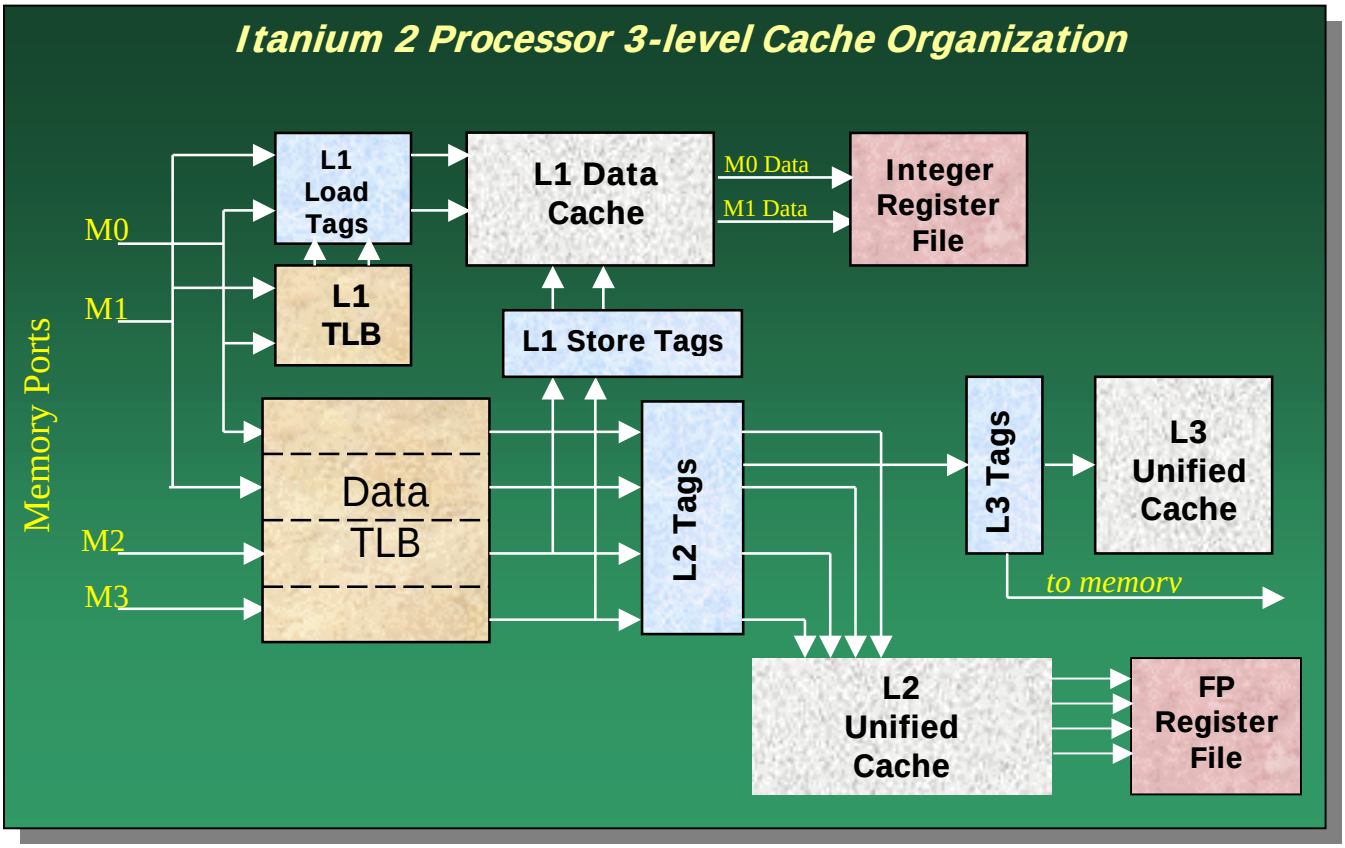
Improved Instruction Prefetching

Feeding the Itanium 2 processor pipeline with 6 instructions per cycle is a monumental task for the instruction fetch unit. Prefetching too much, or prefetching the wrong information can actually degrade performance. In an attempt to provide efficient instruction prefetching, the Itanium 2 processor incorporates hardware to use architecture hints for prefetching. The compiler can direct the Itanium 2 processor when to prefetch and how far to prefetch using several techniques.

- **Demand Prefetching.** When a demand fetch is made the Itanium 2 processor will automatically prefetch the next consecutive L1i cache line, if that extra line is contained within the L2 cache line.
- **Streaming Prefetch.** The compiler can add a hint to any branch instruction (br.many) instructing the Itanium 2 processor to begin prefetching instructions at that address. Up to four requests will be made immediately on this path, and more will be made until a specific condition is met to stop prefetching.
- **Hint Prefetch.** Two other instructions, branch predict (brp) and move to branch register (mov br), can be used by the compiler to trigger limited prefetching. These can prefetch only up to 16 bundles of instructions. This is a good method to use when prefetching must be limited in length.

High Performance 3-level Cache Hierarchy

The Itanium 2 processor cache hierarchy provides significant improvements over the original Itanium processor. The Itanium 2 processor 3-level cache hierarchy is designed for demanding high-end applications. A fast L1d cache provides the single cycle latency needed for integer applications. The L2 cache, with its larger size and high bandwidth, is tuned for data streaming, such as is needed in technical floating-point applications. The large on-chip L3 cache supports the needs of transaction processing and in general the needs of high-end technical and commercial servers.



The Itanium 2 processor L1 data cache is four-ported, allowing two loads and two stores to be executed each cycle. The L1 data cache has a single cycle latency that is enabled by a new cache tag design called a prevalidated cache tag. Each cycle the L1 data cache can deliver up to two integer loads to the integer register file.

Access to the L2 begins at the start of the memory pipeline to provide the fastest possible L2 latency. Each memory access is translated from virtual to physical addressing in the Data TLB. These four translations are done in parallel. The L2 tag lookup is begun immediately, even before a hit or miss is determined in the L1d cache. The early L2 tag lookup allows a quick bypass of the request to the L3 cache, if it is not found in the L2 tags.

The L2 cache is large and is the lowest level cache that contains floating-point data. Four floating-point memory load operations can be delivered from L2 to the floating-point register file each cycle. This bandwidth can completely support 2 floating-point operations per cycle. The L2 also serves as an overflow (2^{nd} level) cache for the L1i instruction cache.

The power of the Itanium 2 processor cache system can be seen in several categories when compared with the original Itanium processor and others:

- **Itanium 2 processor cache latencies are half** those of the original Itanium processor for almost all caches—L1i, L1d, and L3.

LATENCIES	Itanium	Itanium 2
L1i	2 cycles	1 cycle
L1d	2 cycles	1 cycle
L2 (I,FP)	6,9 cycles	5,6 cycles
L3 (I,FP)	21,24 cycles	12,13 cycles

- **Itanium 2 processor's addressing is expanded**—the Itanium 2 processor supports larger virtual and physical addresses, needed by large commercial and technical applications.

ADDRESSES	Itanium	Itanium 2
Virtual Address	50 bit	64 bit
Physical Address	44 bit	50 bit

- **Itanium 2 processor's cache line size is doubled** throughout – this provides an implicit prefetching that reduces cache misses and effective memory latency.

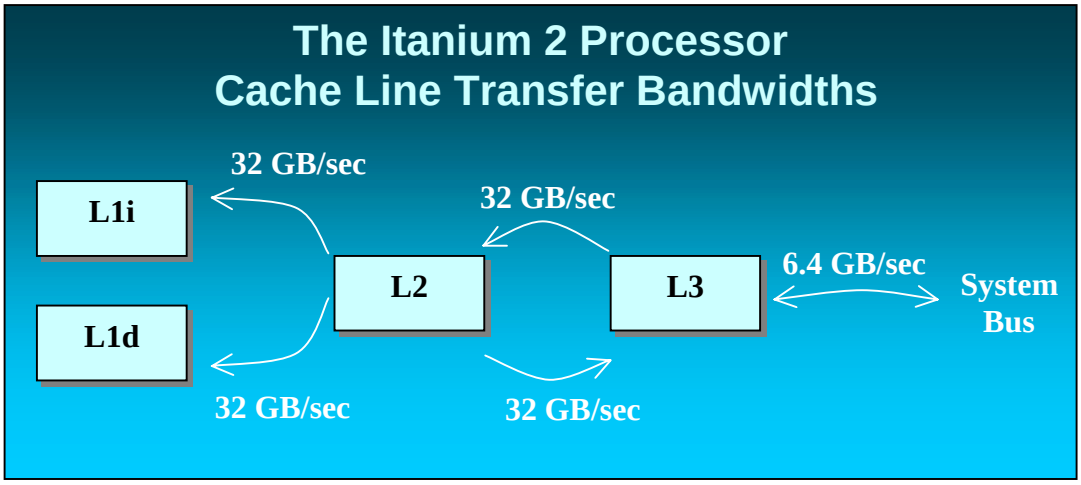
LINE SIZE	Itanium	Itanium 2
L1i	32 bytes	64 bytes
L1d	32 bytes	64 bytes
L2	64 bytes	128 bytes
L3	64 bytes	128 bytes
Bus	64 bytes	128 bytes

- **The Itanium 2 processor has the largest total on-chip caches of all 0.18 micron processors.**

TOTAL ON-CHIP CACHE SIZE	
Itanium 2	3360 KB
Sun UltraSparc III	96 KB
Intel Pentium 4	276 KB
HP PA8700	2304 KB
Alpha 21264B	128 KB
IBM Power 4	1632 KB

- **The Itanium 2 processor has complete MMF bundle support**—the Itanium 2 processor's data cache can support streaming of 4 FP loads and 2 FP stores per cycle to provide high-end technical performance (see *Floating-point Intensive Technical Applications* in the architecture section for a real-world application).

- **The Itanium 2 processor supports directory-based coherency systems**—as systems become larger, directory-based coherency becomes desirable. To support directory systems, the Itanium 2 processor provides a "clean line cast-out" signal to update the directory without transferring any data on the data bus.
- **The Itanium 2 processor supports up to 4GB page sizes**—this allows mapping of large databases and datasets for high-end applications. The original Itanium processor supports up to 256 MB pages sizes.
- **The Itanium 2 processor has doubled fill bandwidth throughout the cache hierarchy**—bandwidth from the system bus to the L3, the L3 to the L2, and the L2 to the L1s are doubled compared to the original Itanium, allowing faster cache line transfers.



Enhanced Speculation Support

Control and Data Speculation are an architectural feature of EPIC to reduce the effective memory latency. These features can be supported at various performance levels by the processor microarchitecture. The Itanium 2 processor has chosen to extend support for speculation in two ways.

First, the Advanced Load Address Table (ALAT) on the Itanium 2 processor is designed to be fully associative (rather than set associative as on Itanium). This increases the effectiveness of the ALAT, providing more reliable performance for compiler optimizations.

Second, the check instructions (chk.a and chk.s) have been optimized to branch directly to recovery code when a speculation fault is detected. The original Itanium processor will first trap to a Speculation Fault trap handler that will then branch to the recovery code. Improving the fault recovery on the speculation check instructions allows the compiler to use speculation in more situations because the fault penalty is reduced. Increasing the amount of speculation will improve performance.

IA-32 Engine Improvements

The Itanium 2 processor's IA-32 translation hardware engine is an enhanced and in-order version of the original Itanium's IA-32 translation engine. It provides x86 compatibility with Katmai (Pentium 3) and earlier processors. It will execute IA-32 applications and I/O drivers directly.

An important advance in the Itanium 2 processor that allowed a simplification from an out-of-order to an in-order IA-32 translation engine was the design of a one-cycle LAGEN unit. The LAGEN unit forms the address for IA-32 instructions. On the original Itanium processor, this function takes 2 cycles.

The performance of the Itanium 2 processor's IA-32 engine is expected to be comparable with a 300 MHz Pentium Pro.

Hardware Performance Monitor Aids Performance Tuning

The Itanium 2 processor includes special performance monitor hardware to enable software to optimize application and system performance. (This capability is even more robust than is available on PA-RISC.) Dynamic optimization software can use this information in real-time to optimize performance as the characteristics of the application change dynamically.

Features of the Itanium 2 processor performance monitor are:

- over 350 distinct events may be counted
- four 47-bit generic performance counters are available
- two opcode match registers can be used to select specific instruction types
- Instruction Event Address Registers (EAR) capture the addresses of selected instruction cache or instruction TLB misses
- Data Event Address Registers capture the address of selected data cache or data TLB misses
- Branch Trace Buffer contains 8 entries for capture of branch information

Enhancements for PA-RISC Customers

Current HP PA-RISC customers can look forward to many benefits in the new Itanium 2 processor systems. To name a few…

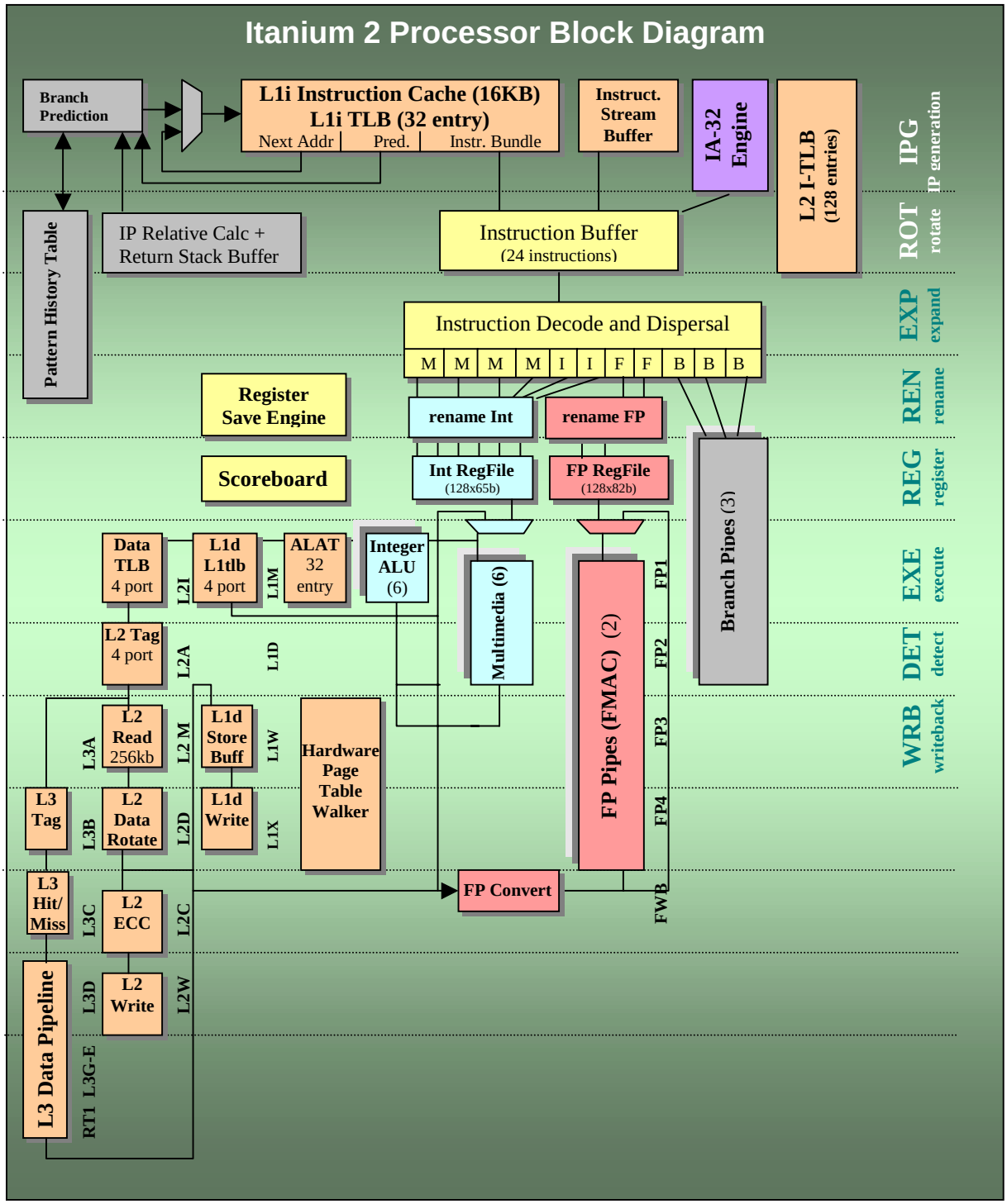
- More instructions per cycle maximum issue rate (6 vs. 4)
- 3 OS capabilities (HP-UX, Linux, Windows)
- Ability to execute IA-32 code
- Larger on-chip caches
- Hardware page walker to handle TLB misses
- More extensive performance monitor for compiler optimization
- Industry Standard Architecture
- 128 64-bit integer registers
- 128 82-bit floating-point registers
- Enhanced Multimedia instruction set
- Control & Data Speculation (advanced loads)
- Predication of instructions

ITANIUM 2 PROCESSOR CACHE SPECIFICATIONS

Characteristic	L1i	L1d
CONTENT:	Instructions Only	Data Only
SIZE:	16KB	16 KB
ASSOCIATIVITY:	4 way	4 way
MIN. LATENCY:	1 cycle	1 cycle (integer only)
LINE SIZE:	64 bytes	64 bytes
ORGANIZATION:	8 “groups”	8 “groups”
MEMORY PORTS:	dual port (1R + 1W)	4 ported (2 read + 2 write)
MEMORY INDEX:	physical	physical
READ DATA PATH WIDTH:	256 bits	2 x 64 bits
WRITE DATA PATH WIDTH:	NA	2 x 64 bits
TAG DESIGN:	Prevalidated Tag	Prevalidated Load Tag, Conventional Store Tag
TAG PORTS:	1 read + 1 write	2 read + 2 write
STORE POLICY:	NA	Write-through
CACHE ALLOCATION POLICY:	On cacheable miss	On cacheable integer load miss or prefetch, NOT on writes
REPLACEMENT:	LRU	NRU (not recently used)
ORDERING:	In order	In order
MISSES:	8 outstanding fetches	8 outstanding misses
FILL TRANSFER RATE:	32 bytes/cycle	32 bytes/cycle
FILL TO MEMORY:	1 cycle	1 cycle

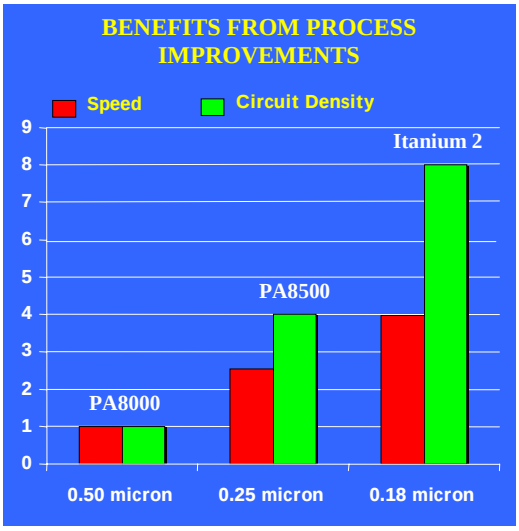
Characteristic	L2	L3
CONTENT:	Unified: I + D	Unified: I + D
SIZE:	256 KB	3 MB
ASSOCIATIVITY:	8 way	12 way
LATENCY:	5, 7, 9 cycle (Integer), 6, 8, 10 cycle (FP), 7, 9, 11 cycle (instruction)	12, 16 cycle (Integer), 13, 17 cycle (FP), 14, 18 cycle (instruction)
LINE SIZE:	128 bytes	128 bytes
ORGANIZATION:	16 banks	1 bank
MEMORY PORTS:	pseudo 4 read or write	1 read or write
MEMORY INDEX:	physical	physical
READ DATA PATH WIDTH:	4 x 82 b (FP) + 2 x 64 bit (int) actually 4 x 128 bits	1 x 32B (to L2)
WRITE DATA PATH WIDTH:	2 x 82 bits	1 x 32B (from L2)
STREAMING OF DATA:	Simultaneous streaming of read and write data to int/FP	Simultaneous streaming of read and write data to L2
TAG DESIGN:	Conventional Tag, early tag	Conventional Tag, early tag
TAG PORTS:	4 read or write	1 read or write
STORE POLICY:	Write-back	Write-back
CACHE ALLOCATION POLICY:	On any cacheable L2 miss	On any cacheable L3 Miss except L2 victim writeback
REPLACEMENT:	NRU (not recently used)	NRU (not recently used)
ORDERING:	Out of order issue/complete	Out of order issue/complete
REQUEST-IN-PROGRESS QUEUES:	48 total (32 + 16 fill requests)	
MISSES:	Non-blocking, 16 misses	Non-blocking, 16 R + 6 W
FILL TRANSFER RATE:	32 bytes/cycle	32 bytes/cycle
FILL TO MEMORY:	1 cycle	4 cycles

Characteristic	L1d TLB	Data TLB (D-TLB)
USE:	L1 integer load translates	L1 protection check and store translations, all L2, L3, memory translates
CONTENT:	Integer loads	All Integer/FP TLBs
SIZE:	32 entries	128 entries
ASSOC:	fully associative	fully associative
LATENCY:	<1 cycle	1 cycle
PAGE SIZES:	4KB	4KB to 4GB
PORTS:	2 read or 1 write	4 read or 1 write
ALLOCATE:	integer load miss	on any DTLB miss


Itanium 2 Processor Block Diagram
Branch Prediction
L1i Instruction Cache (16KB)
L1i TLB (32 entry)
Next Addr
Pred.
Instr. Bundle
Instruct. Stream Buffer
IA-32 Engine
L2 I-TLB (128 entries)
IPG
IP generation
ROT
rotate
Pattern History Table
IP Relative Calc + Return Stack Buffer
Instruction Buffer
(24 instructions)
EXP
expand
Instruction Decode and Dispersal
M M M M I I F F B B B
REN
rename
Register Save Engine
rename Int
rename FP
Scoreboard
Int RegFile
(128x65b)
FP RegFile
(128x82b)
REG
register
Branch Pipes (3)
Data TLB 4 port
L2I
L1d L1tlb 4 port
L1M
ALAT 32 entry
Integer ALU (6)
Multimedia (6)
FP Pipes (FMAC) (2)
FP1
EXE
execute
L2 Tag 4 port
L2A
L1D
FP2
DET
detect
L2 Read 256kb
L2 M
L1d Store Buff
L1W
Hardware Page Table Walker
FP3
WRB
writeback
L3A
L3 Tag
L3B
L2 Data Rotate
L2D
L1d Write
L1X
FP4
L3 Hit/ Miss
L3C
L2 ECC
L2C
FP Convert
FWB
L3 Data Pipeline
L3D
L2 Write
L2W
RT1
L3G-E

Physical Features

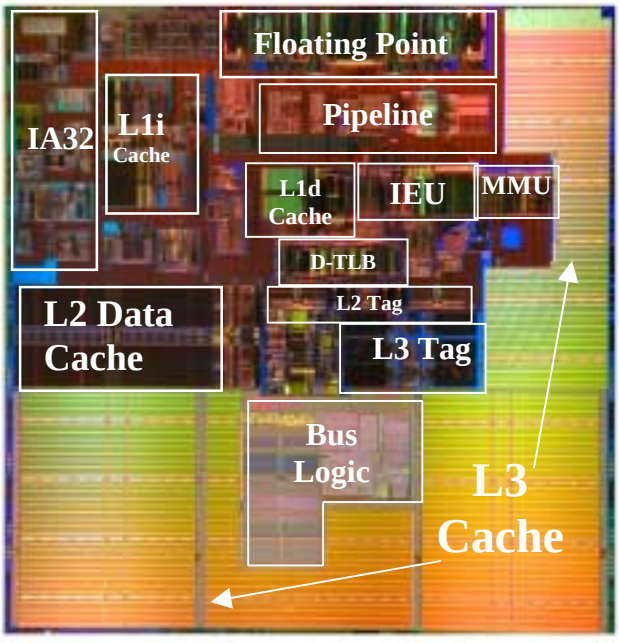
Processor Design Details



The Itanium 2 processor was designed in the Intel P858 0.18 micron CMOS process. As seen from the chart to the right, this allows a 2x improvement in gate density from the previous technology. This new gate density was instrumental in allowing the inclusion of an L3 cache on chip. The first Itanium processor needed to have external memory devices to supply its 4MB of L3 cache, but on the Itanium 2 processor, through advanced memory design techniques, the designers were able to fit 3MB of L3 cache on the processor chip. In addition to lower system cost, this also provided a much lower average access time and much higher bandwidth to the L3. Note from the Itanium 2 processor die photo below that the L3 cache takes up almost half of the die area. As process technology advances, the L3 cache size can increase further because of future improvements in gate density.

The gate density of 0.18 micron technology allowed more to be included in the processor core. There are three caches in the core (L1i, L1d, L2) as well as two large register files (integer and floating-point) with 128 registers each. Two floating-point units that can each deliver a FP multiply-add, 6 integer units, 6 multimedia units, and 4 load-store cache pipelines are provided in the core. Space was available to even include an IA-32 hardware execution unit.

The speed of the new process technology, as well as aggressive circuit design techniques, were key to achieving the 1 GHz processor frequency.

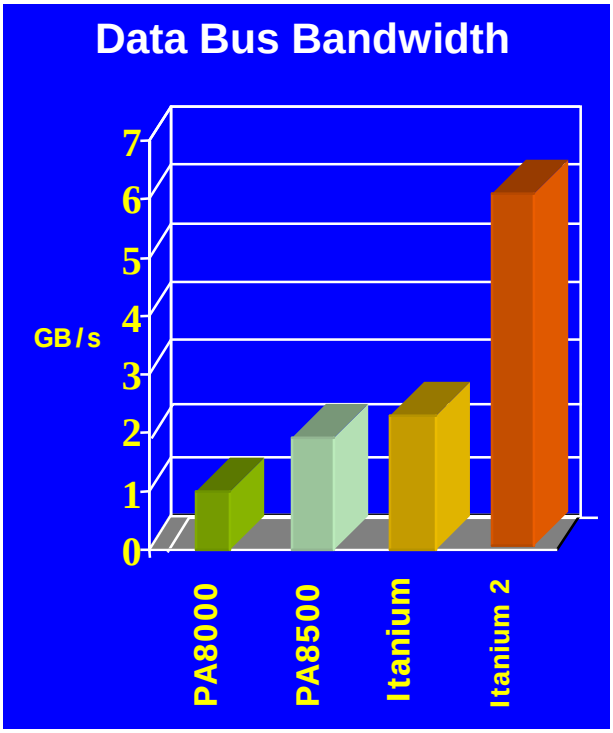


Itanium 2 Processor Specifications

Target Frequency	1 GHz (high bin) 900 MHz (low bin)
IC Process	Intel CMOS
Gate Length	0.18 micron
Metal	6 layer aluminum metal
Die Size	421 mm^2 (full chip) 142 mm^2 (processor core)
Number of Transistors	221 million
Supply Voltage	1.5v @ 1 GHz
Power	130 watts (max) @ 1GHz
Low Power Control	Software controlled processor low power mode
Clocking	on-chip PLL
IO Clock Frequency	200-300 MHz data double pumped to give 400-600 MT/s

System Bus Details

The Itanium 2 processor system bus (also referred to as the Front Side Bus—FSB) provides a 3x improvement in data bandwidth over the first Itanium processors' system bus. The Itanium 2 processor system bus doubles the data path width and increases the bus frequency. The result is a **6.4 GB/sec** maximum data bandwidth at 200MHz. The The Itanium 2 processor system bus also uses a cache line size that is twice that of the first Itanium system bus, bringing more data to the processor for every bus request. These enhancements benefit applications that need sequential data or instruction streams (like many technical applications) and process startup events. Doubling the cache line size works like an automatic sequential data prefetching function.



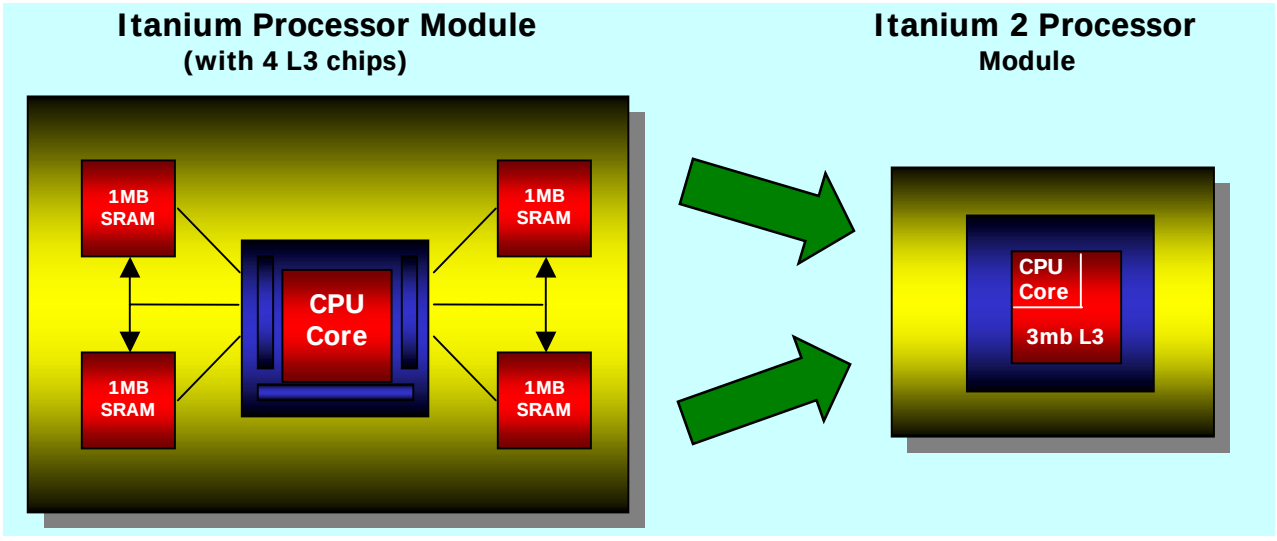
Itanium 2 Processor System Bus Specifications

	Itanium	Itanium 2
Bus Type	Split Address/Data	Split Address/Data
Data Bus Size	64 bits of data + ECC = 72 bits	128 bits of data + ECC = 144 bits
Double-pumped data bus?	Yes, 266 MT/sec	Yes, 400 MT/sec
Data Bus Line Size	64 bytes	128 bytes
Data Bus Bandwidth	2.1 GB/sec @ 133 MHz	6.4 GB/sec @ 200 MHz
Bus Ratios supported	2:n where 8 <= n <= 16	2:n where 8 <= n <= 23
Bus Frequency	133 MHz (with 4 CPUs)	200 MHz (with 4 CPUs)
Processors Supported/bus	up to 4	up to 4

Package and System Details

The Itanium 2 package is different from the package used for the first Itanium processor. The Itanium 2 processor package is called the PAC611. In addition to the extra data bus pins for the Itanium 2 processor system bus, the package does not need to support multiple chips. The original Itanium processor had external L3 memory chips that needed to be mounted, cooled and interconnected in the package. With Itanium 2 processor there is only one chip for the processing unit and its cache. This enables a simpler and less expensive packaging solution.

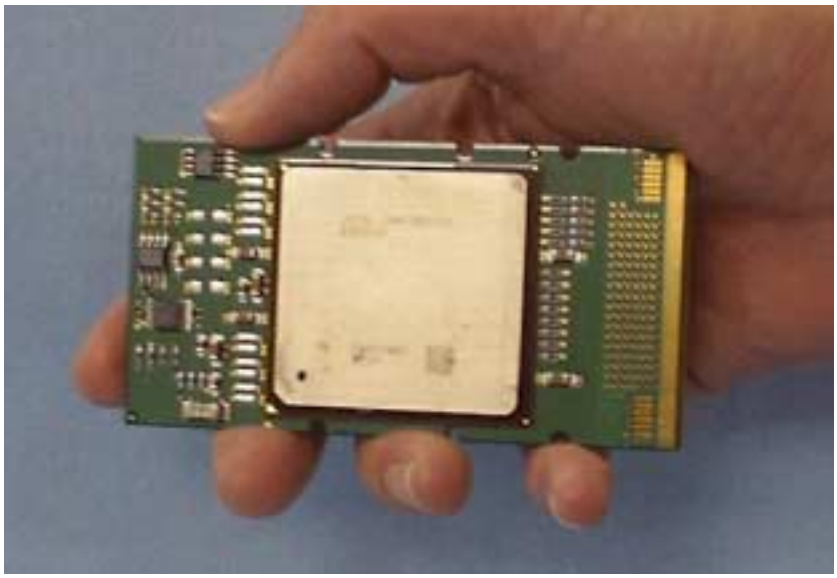
From 1 to 4 Itanium 2 processor chips can be connected to a common system bus to build a "glue-less MP" system of up to 4 ways. For enterprise systems, the Itanium 2 processor contains cache functions that support system memory directory coherency. Other system bus functions are available for writing back dirty cache lines. This is useful for check-pointing memory for highly reliable computing.



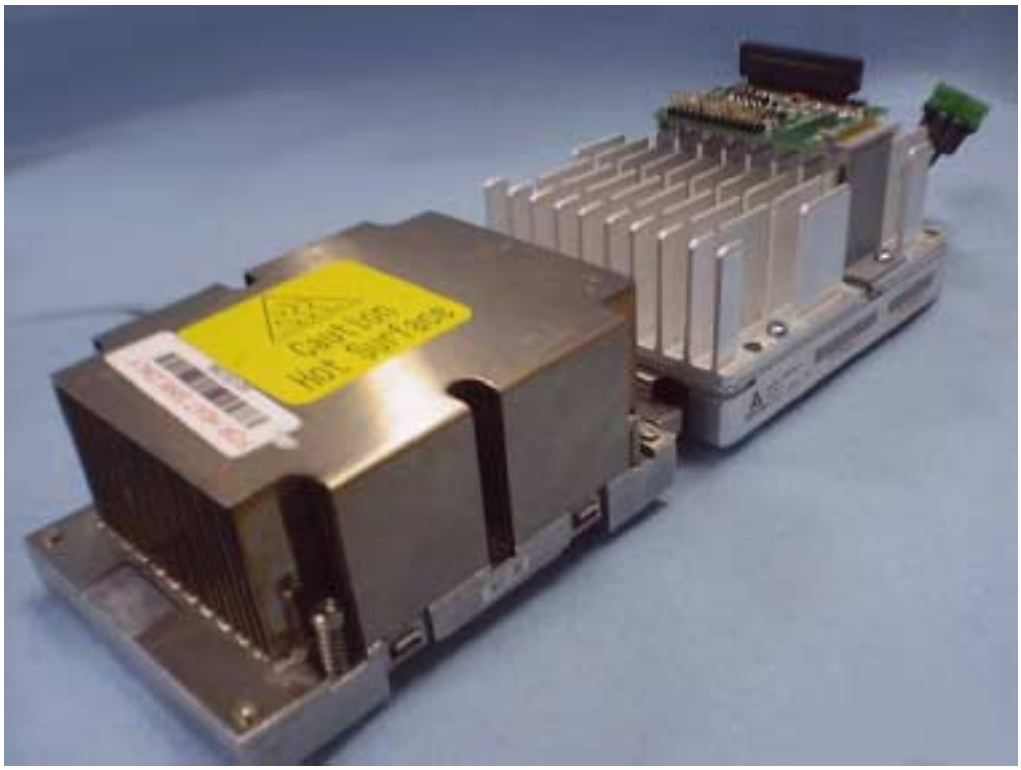
Itanium 2 Package and System Specifications

	Itanium	Itanium 2
Package Type	PAC418	PAC611
Intel Chipset	460GX	870
Packaging	C4 die attach OLGA organic land grid array	C4 die attach OLGA organic land grid array
Logic die/package	5 chips	Single-chip
Back-side Bus	Yes, 12.8 GB/sec	No
L3 memory parts	4 1MB die	none
Bus Frequency	133 MHz (with 4 CPUs)	200 MHz (with 4 CPUs)
Glueless MP	Up to 4 way	Up to 4 way
Directory Support	no special features	yes
System Line Size	64 bytes	128 bytes
Max Physical Adrress	44 bits	50 bits
Max Virtual Address	50 bits	64 bits
Max Page Size Supported	256 MB	4 GB
In-Order-Queue Size	8 entry	8 entry
Deferred Queue Size	16 entry	18 entry

Itanium 2 Processor Module



Itanium 2 Processor Module with Heatsink (in front) and Power Pod (behind)



Reliability, Availability, Serviceability and Maintainability

Always On Computing

Hewlett-Packard is committed to delivering an "always on" computing environment to the customer. This is not only a desire, it is a demand of our customers. Many customers run mission-critical applications. They can't afford downtime because it translates to lower productivity, higher costs, missed sales, or dissatisfied customers.

"Always On" computing doesn't just happen—it implies new ways to solve problems. It demands higher quality in the hardware and software, new recovery techniques when things do fail, and methods to monitor and control these processes seamlessly.

Many RAS (Reliability, Availability and Serviceability) features are built into the system chipsets and overall system design (e.g. redundant power supplies and fans or "hot swap" capabilities)[3]. Other RAS features rely on software or firmware routines to help recover from failures. This chapter will focus on the contribution that the Itanium 2 processor makes to improving system reliability and availability.

It would be nice to prevent all hardware failures, but that is not possible. Each new generation of VLSI technology moves to smaller circuit sizes and lower operating voltages. This increases the probability of processor failures. As a result, designers must work harder on each new processor generation to provide higher levels of reliability.

To aid our discussion on hardware failures, we can simplistically classify processor failures into three groups:

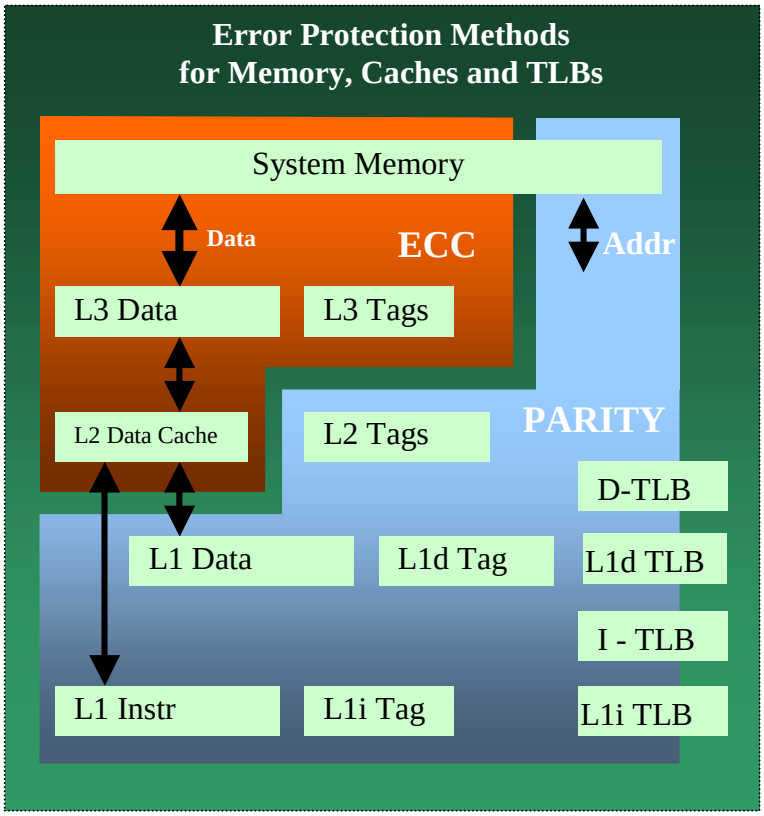
IN BIRTH:	**marginal design, manufacturing variance, or inadequate testing of a processor chip**
IN LIFE:	**failures due to information changing in the processor due to a radiation particle hit**
IN OLD AGE:	**failures due to aging of devices, fatigue in mechanical connections and external factors**

Preventing Failures in Customer Systems

It is possible to *prevent* failing parts from getting into systems by solving the "baby chip " failures. All Itanium Family Processors are produced in Intel fabs. Intel is a leader in semiconductor manufacturing and has significant experience in delivering quality parts to its customers. Intel controls the process technology and defines the design rules for Itanium processors. Based on the Intel design quality rules, design-for-test (DFT) methodology and their manufacturing and testing experience, extremely high quality parts are expected for the Itanium family.

Correcting Hardware Failures Transparently—Withstanding Nuclear Particles!

During the lifespan of a good processor chip, failures can be generated by external forces. Most troubling are strikes by external sources of ionizing radiation. These may be either alpha particles (a result of radioactive decay) or high-energy neutrons (which originate in deep space). Either of these can hit an integrated circuit device and cause the data held by that device to change. These particles can hit logic circuits. Luckily many times these strikes cause no harm to the operation, since they may hit a device that is sitting idle (like an empty pipeline stage), or maybe one that is irrelevant to the current operation (a floating point multiply result is changed, but an integer add was the operation being performed). Other strikes in logic may cause the processor to "hang" and a watch-dog timer may need to detect this failure and trigger a recovery operation. Random logic failures are hard to detect without duplicating logic. The good thing is that random logic failures are statistically a small percentage of the failures in a processor.



The major problem area for processor circuit failures is related to memory structures. This is for three reasons: (1) they are dense so it is easier for a particle to hit something important that will cause a problem, and (2) they are used to hold data for a relatively long period of time, and for the Itanium 2 processor, (3) they make up a large portion of the die area. Changing that data will more frequently cause a data integrity problem. In the diagram to the right the protection schemes for the memory, data buses, data and instruction caches, cache tags, and TLBs (translation look-aside buffers) are illustrated as implemented on the Itanium 2 processor.

For the larger memory and cache structures an Error Correcting Code (ECC) is implemented which will allow a Single bit Error to be Corrected and a Double bit Error to be Detected (SECDED). This covers the system memory, the L3 and L2 unified caches, the L3 tags and the buses that interconnect them. Any single bit failure in this area can be corrected automatically by the Itanium 2 processor hardware, transparent to the firmware, OS or application software. Because memory cells are very small in the Itanium 2 processor, it is possible for a single radiation strike to impact multiple bits of a memory array. To reduce the probability of generating more than one error on a single ECC protected field, the memory array is interleaved so that a single strike can at most damage a single bit in multiple ECC protected fields. This technique improves the probability that a failure can be corrected transparently by the hardware. When multiple errors are detected, the bad data is flagged as "poisoned"—this feature will be described later.

Many of the other structures have *parity protection*. While parity protection doesn't allow correction, the interaction of the processor and firmware can allow transparent recovery. For example, in the cases of the L1i and L1d caches, their data errors can be detected before process state is damaged. In case of the L1i, an instruction parity error is detected before the instruction is ever executed and the hardware automatically re-fetches a new copy of the instruction from L2. In L1d a parity error on a memory load will be turned into a miss and re-loaded from L2. Failures in L1d tags or stores can be recoverable, since the L1d is a write-thru cache, it has no data that isn't held elsewhere. Some L1d failures may signal firmware to invalidate the contents of the L1d cache and resume processing without any damage to the user's application. The L2 tag is covered by parity and needs a firmware recovery from a fault, but the L2 state control bits (MESI) use a special technique that allows automatic correction of a failure whenever the "modified" bit is corrupted. Since modified data may only exist in the L1 cache, this feature can prevent corruption of memory and eventual application failure.

Other *performance-enhancing structures*, like branch prediction memories, have no hardware protection, but also will not corrupt a process in execution. In the case of corruption of the branch prediction memory by an alpha hit, the branch prediction made by reading this data will be incorrect, but the penalty is a normal branch misprediction pipeline resteer of a few cycles. Eventually the bad data will be overwritten, and the performance advantages will resume. Similarly, failures in the replacement algorithms of the caches do not cause data corruption, but merely a lowering of cache efficiency until the bad replacement data is overwritten.

Containing Hardware Failures to Minimize Impacts

If a hardware failure cannot be transparently corrected by the processor, the processor hardware, firmware (PAL and SAL), and the OS work together to contain the failure, minimize the impact and resume normal processing without bringing down any more of the processing system than is needed. The processor may notify the firmware, and the PAL firmware can trigger local fix-up operations. Alternatively, the firmware may defer the repair operation to the OS through the SAL.

Poisoned data, discussed earlier, is a result of a multiple bit error detected by ECC circuitry. Since it can't be transparently repaired, the processor indicates, by a special control means, that the data is not valid and therefore cannot be used. Sometimes that data will never be accessed again (such as a line that has been read for its data already, or an instruction cache line that has already been executed), in these cases the poisoned data cache line will eventually be overwritten by newer cache data, and the user's task will be unaware of the problem. Of greater concern is a poisoned cache line which must be read, updated, or written back to the system memory. When a poisoned line is actually used, the processor must decide what action to take. The rules are:

Attempted use of a Poisoned Cache Line	Resulting Action
On a data load or instruction fetch:	trigger a local Machine Check Abort (MCA)
On a store:	no action
Bus Snoop or eviction causing a Writeback:	writeback cache line with double-bit ECC error

Using this method only the processes that use the poisoned data will be terminated; the remainder of the system can continue processing unaffected.

Hardware Error Containment. In addition to poisoning bad data, the Itanium 2 processor hardware builds in many other techniques to detect *and contain* hardware failures before a hardware failure can

damage the user's application. In most cases, hardware failures are detected and signal the error to receiving units in time to prevent damaging the processor state (i.e. writing bad data to caches or registers). Containment of the failure can help prevent damage from migrating to other applications, and can also improve the ability to recover from errors using firmware or software techniques.

Watchdog Timeout Timer and other fatal MCAs (machine check aborts) pass recovery control on to the firmware. The Watchdog Timeout Timer is provided to protect against processors that are "hung". The processor tracks the time that the pipeline has been stalled and counts up for each stalled cycle until a limit has been reached. When the limit is reached this error is passed on for firmware recovery.

Faults presented to the firmware are of three major types, so that errors can be processed with the minimum impact to the system. These general classes are:

Fault Type	Action
Local Machine Check Abort:	Only the local processor enters PAL recovery code
Global Machine Check Abort:	All processors on the system bus network enter the PAL recovery code
Hardware bus reset:	A hardware reset is performed to ensure forward progress and then all processors on the system bus network enter the PAL recovery code

Serviceability/Maintainability

Error logging allows the Information Technology administrator to monitor the level of activity of correctable and uncorrectable failures, and also trace the source of failures in a large system. Error logging, with firmware support, will track:

- the type of error (corrected, local Machine Check Abort, global Machine Check Abort, bus reset)
- the status (valid, overflow)
- the physical address on the system bus
- the bus transaction type and size

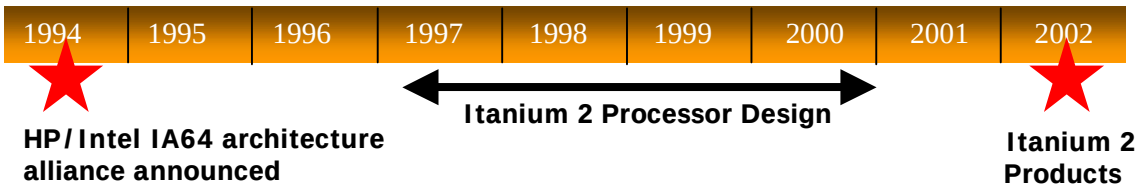
Firmware can handle multiple Machine Check Aborts through the use of an overflow bit that is set by hardware when logging data may be lost.

Other debugging hardware is provided on the Itanium 2 processor to allow special case debugging and recovery operations.

- Internal debug triggering capabilities (capture information on a specific event)
- Logging data to internal registers to be read out later
- Execution trace logging via a buffer of all taken branches

History and Future Plans

In 1997 the Itanium 2 processor design team finalized the project goals and targeted the processor design for the Intel P858 (0.18 micron) technology. The Itanium 2 design effort then diverged from the work being done on the original Itanium processor. This was a joint project staffed with engineers from HP Ft. Collins and various Intel sites, using development tools from both companies. The design was finished and ready for the initial "tape release" in December of 2000. After samples were received and with three weeks of debugging, HP-UX (UNIX), Windows 2000, and Linux systems booted on Itanium 2 systems with an HP-designed chipset.



What Others Have Said…

The Itanium 2 processor design has been anticipated as a major event even before its release:

> *"…the second generation IA-64 microprocessor will simply knock your socks off."*
> Fred Pollack, Intel Fellow
> Microprocessor Forum, 1997

The demonstration of three HP systems at the Intel Developers Conference (Feb, 2001), each running a different operating system (HP-UX, Windows, Linux) on the Itanium 2 processor, brought about even more acclaim from Intel and industry observers:

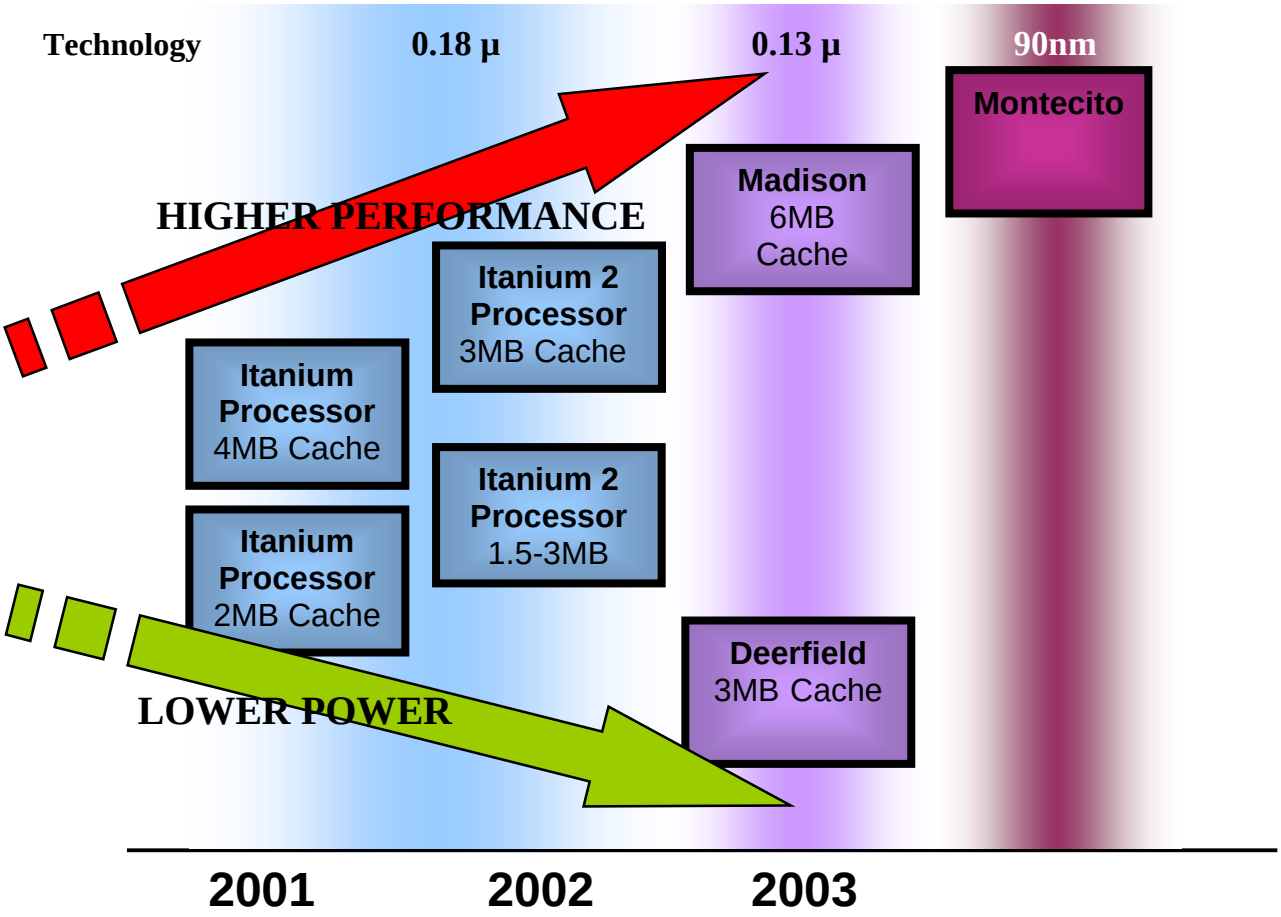
> *"this level of functionality on an A0 stepping in our history is really unprecedented."*
> Paul Otellini
> Intel Corp., 2001

> *"That's great progress for them to make after three weeks of silicon,"*
> Linley Gwennap
> Principal Analyst at the Linley Group

At the public release of the Itanium 2 processor on July 8, 2002 analyst Brad Day of the Giga Information Group said:

> *"Itanium has now crossed the threshold to compete on par and in some cases to compete with an advantage…It wasn't until release of benchmarks by HP and Microsoft around SAP and SQL Server 2000 that we believed the Itanium 2 architecture had the balance to be aggressive both on technical and commercial computing."*
> *quoted by Stephen Shankland in "HP Touts Advantages of Itanium 2", CNET, 7/8/02*

What About the Future of the Itanium Processor Family?



While the Itanium 2 processor represents a major step in the Itanium Processor Family, work has already begun on the next generation of processors. Since the Itanium 2 processor was designed in the Intel P858 process, late in that process's life, it is reasonable to move the Itanium 2 microarchitecture into the latest technology. That enhancement will allow generation of both lower power and higher performance products. That is the goal for the next-generation Madison and Deerfield products. Madison will increase performance by increasing frequency and the L3 cache size, and Deerfield will provide a lower power alternative.

References

GENERAL ITANIUM INFORMATION

[1] Inside *Intel's Itanium: A Strategic Planning Discussion*, March 2000, Aberdeen Group, Inc, Boston MA. See http://developer.intel.com/software/idap/media/pdf/tools/eapps/eb3_servr_pdf_07.pdf.

Intel Itanium 2 Processor Reference Manual for Software Development and Optimization, http://developer.intel.com/design/itanium.

Web sources: http://www.hp.com/go/itaniumdeveloper - HP Itanium developer info
http://www.cpus.hp.com - HP Itanium architecture, processor design info
http://developer.intel.com - Intel Itanium developer info

ITANIUM ARCHITECTURE

[2] "*EPIC: Explicitly Parallel Instruction Computing*", Michael S. Schlansker and B. Ramakrishna Rau, *Computer*, Feb 2000, 37-47

EPIC: An Architecture of Instruction Level Parallel Processors, M. Schlansker, B. Rau, HP Laboratories Report HPL-1999-111, February 2000

Intel® Itanium™ Architecture Software Developer's Manual,
Volume 1, Application Architecture,
Volume 2, System Architecture,
Volume 3, Instruction Set Reference, Intel Corp., http://developer.intel.com

"*Introducing the IA-64 Architecture*", Jerry Huck et. al., *IEEE Micro*, Sept/Oct 2000, Volume 20, Number 5, 12-23

"OS and Compiler Considerations in the Design of the IA-64 Architecture", Rumi Zahir et. al., ASPLOS IX, 11/2000, 212-221

RELIABILITY, AVAILABILITY, SERVICABILITY

[3] "*High Availability and Reliability in the Itanium Processor*", Nhon Quach, *IEEE Micro*, Sept/Oct 2000, Volume 20, Number 5, 61-69